Downloaded from orbit.dtu.dk on: May 05, 2023

DTU Library

Specifying and verifying requirements of real-time systems

Ravn, Anders P.; Rischel, Hans; Hansen, Kirsten Mark

Published in:
I E E E Transactions on Software Engineering

Link to article, DOI:
10.1109/32.210306

Publication date:
1993

Document Version
Publisher's PDF, also known as Version of record

Link back to DTU Orbit

Citation (APA):
Ravn, A. P., Rischel, H., & Hansen, K. M. (1993). Specifying and verifying requirements of real-time systems. *I E E E Transactions on Software Engineering*, *19*(1), 41-55. https://doi.org/10.1109/32.210306

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Specifying and Verifying Requirements of Real-Time Systems

Anders P. Ravn, *Member, IEEE,* Hans Rischel, *Member, IEEE,* and Kirsten Mark Hansen

Abstract— **An approach to specification of requirements and verification of design for real-time systems is presented. A system is defined by a conventional mathematical model for a dynamic system where application specific states denote functions of real time. Specifications are formulas in duration calculus, a real-time interval logic, where predicates define durations of states. Requirements define safety and functionality constraints on the system or a component. A top-level design is given by a control law: a predicate that defines an automaton controlling the transition between phases of operation. Each phase maintains certain relations among the system states; this is analogous to the control functions known from conventional control theory. The top-level design is decomposed into an architecture for a distributed system with specifications for sensor, actuator, and program components. Programs control the distributed computation through synchronous events. Sensors and actuators relate events with system states. Verification is a deduction showing that a design implies requirements.**

Index Terms— **Real-time systems, requirements engineering, specification, verification.**

I. Introduction

An engineer who is designing an embedded computer system must have a deep insight in the properties of the controlled physical processes. When the system is safety critical, it becomes particularly important that this insight be made explicit and forms the basis for design. This was already observed by Heninger [12] in connection with the A7 flight program, and the viewpoint is pursued in later work by Parnas, for example [32]–[34] and appears also in [18]. It has led to increased focus on requirements engineering [4]. The challenge, however, is to find suitable mathematical theories and notations that allow a designer to record such insight. It is also crucial, but often neglected, that the theory shall make it practical for a designer to use mathematical reasoning when checking that a design conforms to the requirements. In our work with case studies within the Provably Correct Systems (ProCoS) project [3] we have studied the problem of specifying and verifying total system requirements [13], [36], [37]. This paper describes the resulting approach to requirements engineering, design, and verification for real-time control systems.

A central property of such systems is that they can be modeled by states changing over time. The theory of dynamic systems, see, e.g., [23], or more specifically control theory, see, e.g., [7], [22], are rather specialized, however. They concentrate on systems that can be described by a single invariant or control law in the form of a differential equation, or in the discrete case a difference equation with a fixed time step. In order to be really tractable, the equations are restricted to be linear, although some results have been reached for nonlinear systems, e.g., [17]. Dynamic systems theory is thus not readily applicable to systems with varying time steps, nondeterministic state changes, or with control laws that depend on modes of operation. Furthermore, the notations are not well suited for composition of interacting systems, see e.g., part XII of [46].

Composition seems the only way of dealing with the complex state spaces that arise whenever programming is involved, thus we were led to investigate logic where the intimate relationship between conjunction and composition of concurrently active systems [16] can be used as a composition principle. The logic should also be able to specify real-time constraints, and not just partial ordering of actions. There are many real-time logics, but they seem to fall into two broad classes: explicit time or implicit time. With explicit time formalisms, time is an ordinary variable, represented by event occurrence symbols as in RTL [19]. The time variable may also represent a time interval, see, e.g., [43], or it may be a variable in a temporal logic [35]. In explicit time logic, timing constraints are encoded as inequations over arithmetic expressions in time variables. Through some experiments we found this approach to be less satisfactory because there is no clear relationship between a natural language formulation of constraints and the resulting inequations. We observed that the timing constraints seem to be formulated as constraints on the duration of critical states. Implicit time logic, e.g., metric temporal logic [21], [35] or ISL [9] recognize this by using temporal operators that constrain the extent of a state. It is, however, not possible to express critical durations of the following form: "Within any period of length T, the critical state S must only occur c % of the time." Such formulations describe a constraint on the sum of an arbitrary number of extents and leads to a duration concept, i.e., a logic where time is observed through the accumulated presence of a state. It was through joint work in the ProCoS project that duration calculus emerged [47]. It is based on interval temporal logic [1], [2], [10], [29],

Manuscript received August 1, 1992. This work was partially supported by the Commission of the European Communities (CEC) under the ESPRIT program in the field of Basic Research Action, project 3104: "ProCoS: Provably Correct Systems," and by the Danish Technical Research Council under the "RapID" program. A previous version of this paper was presented at the ACM Sigsoft '91 Conference on Software for Critical Systems, New Orleans, LA, December 1991. Recommended by N. G. Leveson and P. G. Neumann.

The authors are with the Department of Computer Science, Technical University of Denmark, DK 2800 Lyngby, Denmark.

IEEE Log Number 9205026.

0098-5589/93$03.00 © 1993 IEEE

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on November 27, 2009 at 04:43 from IEEE Xplore. Restrictions apply.

[42] and the duration concept. Duration calculus gives us the means to describe critical durations, progress from state to state, and stability of a state. Furthermore, it has a set of rules that have been useful when reasoning about timing properties.

Concurrently with development of duration calculus, we experimented with ways of applying the theory in a systematic manner to requirements specification, design, and verification. Through many iterations the following approach has emerged:

1) Top-level requirements are specified by constraints on a set of entities X representing the relevant physical states as a function of real time. The constraints are typically a conjunction of formulas. One class of formulas constrain the duration of critical states. Another class of formulas specify progress properties: within a certain time the system should move from one state to another.
2) A top-level design is given by some assumptions and a control law. The assumptions record the intrinsic design of the physical system (the *Nat* relation in [34]). They are requirements that the system imposes on the environment, for instance, that certain states are considered physically impossible or that certain progress properties are ensured by physical processes. The control law consists of two parts:
   a) A finite state machine or automaton describing how control progresses through a number of phases. This is specified by a formula over a phase control state `P`.
   b) A set of phase requirements that for each phase define progress and stability constraints to be satisfied during that phase. A phase requirement determines also whether the phase is stable or control shall enter a new phase.
3) An architecture is defined. It consists of specifications for a set of concurrently operating sensor, actuator and program components that synchronize through events. This distributed system is controlled by a scheduler. The scheduler maintains a trace state `tr`, recording the sequence of events passed between components. In order to ensure that a conjunction of component specifications remain consistent, we cannot allow upper bounds on the duration of a phase (this reflects that a set of synchronized state machines can only move when all are ready). Upper bounds are thus rewritten as readiness to progress, using a private state `Ref` (cf. [16]) for each component. The scheduler ensures progress when all components indicate that they are ready to proceed. The system state X is also distributed as private states for components.

This is an outline of the approach; but we hasten to add that there are still many points that need investigation; we return to some of these points in the conclusion. A similar approach to Requirements Capture and top-level design is found in Parnas' work [34]. Use of finite state automatons to specify designs for safety-critical systems are also investigated in [18] and as a paradigm for fault-tolerant systems in [40]. It is also the basis for the ProCoS program specification language [31] and the Statechart formalism [15]. The decomposition into an architecture is primarily inspired by Roscoe and Reed's work on timed CSP [38], [41], and He Jifeng's work on real-time semantics [20].

Section II summarizes duration calculus. Section III presents the running example and the (informal) requirements. Section IV formalizes system requirements. Section V introduces a top-level design with a control law. The top-level design is verified in section VI. Section VII specifies the architecture. Verification is in Section VIII, followed by the conclusion in Section IX.

II. Specification Language

We use the well-known *time-domain model* of systems and control theory [7], [22], [23] for modeling systems. A system is described by a collection of *states* (often called state variables) that are functions of *time*, modeled by the real numbers.

Properties of systems are expressed by constraining the states over time. We wish to express requirements and design without explicit mentioning of particular time instants, and introduce a notation that is a real-time interval logic based on state durations [14], [47].

A. Syntax

We assume (upper case) names X, Y, ... for the states together with (notation for) their *value domains* $Type_X$, $Type_Y, \ldots$, given in a suitable specification language, such as **Z** (cf. [44]) or **VDM** (cf. [5]). This language must comprise a number of standard data types with operators and constants, including the type $\mathbf{R}$ of real numbers and the type **Bool** of Boolean. We denote the Boolean constants by tt and ff (the names $true$ and $false$ are reserved for duration formulas). We use lower-case names a, b, ..., x, y, ..., to denote *constants* and *variables* of any type. As usual in mathematical logic, a variable is an arbitrary value that may be bound by a quantifier, whereas a constant is fixed in each interpretation.

State Expressions and State Assertions. A *state expression* may be of any type and is generated by

1) Any state, constant and variable is a state expression.
2) Any well-formed expression formed from an n-ary operator symbol and state expressions $S_1, \ldots, S_n$ is a state expression.

A *state assertion* is a state expression of type **Bool**.

Durations and Duration Terms. For any state assertion P, $\int P$ is a *duration* and of type $\mathbf{R}$. A *duration term* is also of type $\mathbf{R}$ and generated by

1) Durations, real constants, and real variables are duration terms.
2) Any well-formed expression formed from an n-ary operator symbol of type $\mathbf{R}$ and duration terms $r_1, \ldots, r_n$ is a duration term.

The symbol ℓ is used as an abbreviation for $\int tt$.

Duration Formulas. Any expression formed from an n-ary predicate symbol on $\mathbf{R}$ and duration terms $r_1, \ldots, r_n$ is an *atomic duration formula*. A *duration formula* is of type **Bool**

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on November 27, 2009 at 04:43 from IEEE Xplore. Restrictions apply.

and generated by

1) Atomic duration formulas and the special symbols $true$ and $false$ are duration formulas.
2) If $\mathcal{D}_1$ and $\mathcal{D}_2$ are duration formulas, so are the expressions $(\neg\mathcal{D}_1)$, $(\mathcal{D}_1 \vee \mathcal{D}_2)$ and $(\forall x)\mathcal{D}_1$, where x is a variable.
3) If $\mathcal{D}_1$ and $\mathcal{D}_2$ are duration formulas, so is the expression $(\mathcal{D}_1\,;\,\mathcal{D}_2)$.

B. Semantics

An *interpretation* $\mathcal{I}$ of our formal system corresponds to a particular execution (run) of the system, where each state X denotes a *function*

$$\mathcal{I}(X) : [0,\infty) \rightarrow Type_X$$

giving the state as function of time from the start $t = 0$, and where each constant a is interpreted as a value $\mathcal{I}(a)$ of appropriate type. For given interpretation $\mathcal{I}$, a *valuation* $\mathcal{V}$ asigns a value $\mathcal{V}(x)$ to each variable. By evaluating expressions for each point of time, the interpretation extends to state expressions, which denote functions of time. A constant a or variable x is hereby interpreted as the function with constant value $\mathcal{I}(a)$ or $\mathcal{V}(x)$.

An *observation interval* (or interval for short) is a closed and bounded interval $[b,e] \subset [0,\infty)$. For given interval $[b,e]$, the duration $\int P$ of a state assertion P denotes the real number

$$\int_b^e \chi_P(t)\,dt$$

where

$$\chi_P(t) = \begin{cases} 1 & \text{for} \quad \mathcal{I}(P)(t) = tt \\ 0 & \text{for} \quad \mathcal{I}(P)(t) = ff \end{cases}$$

which is the measure of the set of points in $[b,e]$ where the interpretation of P has value tt. It is assumed that all state assertions denote integrable functions of time. The interpretation then extends to duration terms and duration formulas on each interval: the interpretation of duration terms and atomic duration formulas is defined on each interval $[b,e]$ by evaluating expressions using the values of subterms on the same interval. The formulas $true$ and $false$ as well as composite duration formulas $\neg\mathcal{D}$, $\mathcal{D}_1 \vee \mathcal{D}_2$ and $(\forall x)\mathcal{D}$ are interpreted on each interval $[b,e]$ in the same way as usual logical formulas in predicate logic, cf. [11].

The interpretation of a "chop" formula $\mathcal{D}_1;\mathcal{D}_2$ on an interval $[b,e]$ use interpretations of the subformulas $\mathcal{D}_1$ and $\mathcal{D}_2$ on *subintervals* of $[b,e]$. It has the value tt iff a "chop" point m $(b \leq m \leq e)$ can be found such that $\mathcal{D}_1$ is tt on $[b,m]$ and D_2 is tt on $[m,e]$.

Validity. A duration formula $\mathcal{D}$ *holds* on the interval $[b,e]$ in the interpretation $\mathcal{I}$ iff $\mathcal{D}$ has value tt on $[b,e]$ for any valuation $\mathcal{V}$ for $\mathcal{I}$.

The formula $\mathcal{D}$ *holds from start* for the interpretation $\mathcal{I}$ iff it holds on any interval of the form $[0,T]$ for the interpretation $\mathcal{I}$.

A duration formula $\mathcal{D}$ is *valid* (a tautology) iff it holds for every interval $[b,e]$ in any interpretation $\mathcal{I}$. (Note that the validity of a chop formula $\mathcal{D}_1;\mathcal{D}_2$ on an interval $[b,e]$ may depend on different chop points m for different interpretations.)

It is sufficient for a formula to be valid that it holds from start for every interpretation $\mathcal{I}$ (this uses that shifting all functions in an interpretation $\mathcal{I}$ a fixed amount of time b yields another interpretation $\mathcal{I}'$, such that $\mathcal{D}$ holds on $[b,e]$ for $\mathcal{I}$ exactly when $\mathcal{D}$ holds on $[0,e-b]$ for $\mathcal{I}'$).

Finite Variability. Interpretation of states has been confined to integrable functions in order to make the concept of duration well defined. In order to have a well-founded induction, we require that interpretation of any state assertion P has *finite variability*: any interval $[b,e]$ can be divided into finitely many subintervals with $\mathcal{I}(P)$ constant on each open subinterval.

C. Specifications and Refinement

A *specification* for a system is a duration formula $\mathcal{D}$. An interpretation $\mathcal{I}$ is said to *satisfy* the specification if $\mathcal{D}$ holds from start for $\mathcal{I}$. For specifications $\mathcal{D}_1$ and $\mathcal{D}_2$ we say, that $\mathcal{D}_2$ is a *refinement* of $\mathcal{D}_1$ if any interpretation satisfying $\mathcal{D}_2$ also satisfy $\mathcal{D}_1$. It follows that $\mathcal{D}_2$ is a *refinement* of $\mathcal{D}_1$ if the duration formula

$$\mathcal{D}_2 \Rightarrow \mathcal{D}_1$$

is valid.

D. Deductions

A primary goal of using mathematical modeling is the ability to *calculate* properties of the model. The calculations for our notation are *deductions*, verifying that the validity of some formulas implies the validity of others.

It is a goal for the work on duration calculus to formalize deductions such that verification is done by calculations without any reference to a semantical model. The work has not yet reached that stage, but a number of useful axioms and deduction rules have been found. They are listed in the Appendix at the end of the paper. It has been proved (cf. [14]) that this formalization of the duration logic is a relative complete extension of real-valued interval temporal logic.

Most verifications for a real-time system consist of case analysis for a moderate number of cases. The individual cases have deductions that are mostly simple calculations. We give manually developed deduction outlines by necessity. The calculations, however, are simple and stereotypic, so there is reasonable hope for assistance from mechanical deduction asistants.

E. Abbreviations

We use standard abbreviations $\wedge$, $\Rightarrow$, $\Leftrightarrow$, $\exists$, and we introduce abbreviations for commonly used duration formulas, with a state assertion P, duration formulas $\mathcal{D}$, $\mathcal{D}_1$, and $\mathcal{D}_2$, and a positive time constant t as shown in Table I. The abbreviations have the following semantics:

$\lceil\,\rceil$	holds on point intervals $[b,b]$
$\lceil P \rceil$	holds on $[b,e]$ if P has value tt almost everywhere in $[b,e]$ *and* if $b < e$
$\Diamond\mathcal{D}$	holds on $[b,e]$ if $\mathcal{D}$ holds on some subinterval of $[b,e]$

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on November 27, 2009 at 04:43 from IEEE Xplore. Restrictions apply.

TABLE I
ABBREVIATIONS

Abbreviation	Formula	Legend
$\lceil\rceil$	$\ell = 0$	Point
$\lceil P \rceil$	$(\int P = \ell) \wedge (\ell > 0)$	Almost everywhere P
$\Diamond \mathcal{D}$	$true; \mathcal{D}; true$	Somewhere $\mathcal{D}$
$\Box \mathcal{D}$	$\neg(\Diamond(\neg \mathcal{D}))$	Always $\mathcal{D}$
$\mathcal{D}_1 \rightarrow \mathcal{D}_2$	$(\mathcal{D}_1; true) \Rightarrow (\mathcal{D}_1 \vee (\mathcal{D}_1; \mathcal{D}_2; true))$	$\mathcal{D}_2$ follows $\mathcal{D}_1$
$\mathcal{D}_1 \sim t \rightsquigarrow \mathcal{D}_2$	$(\mathcal{D}_1; true) \Rightarrow ((\ell \leq t) \vee ((\ell \leq t); \mathcal{D}_2))$	$\mathcal{D}_1$ leads to $\mathcal{D}_2$ in time t

$\Box \mathcal{D}$ holds on $[b, e]$ if $\mathcal{D}$ holds on any subinterval of $[b, e]$

$\mathcal{D}_1 \rightarrow \mathcal{D}_2$ holds on an interval with $\mathcal{D}_1$ holding on an initial subinterval if $\mathcal{D}_2$ holds on some subinterval from the point (if any) where $\mathcal{D}_1$ ceases to hold

$\mathcal{D}_1 \sim t \rightsquigarrow \mathcal{D}_2$ holds on an interval with $\mathcal{D}_1$ holding on an initial subinterval if $\mathcal{D}_2$ starts holding within time t.

The following rules of precedence are used:

first: $\neg$, $\Box$, $\Diamond$
second: $\vee$, $\wedge$, ;
third: $\Rightarrow$, $\rightarrow$, $\sim t \rightsquigarrow$

The logical operators are overloaded: they are used for state assertions as well as duration formulas, but the meaning can be inferred from the type of the operands, e.g., "$\vee$" denotes disjunction for state assertions in "$\lceil P_1 \vee P_2 \rceil$" and disjunction for duration formulas in "$\lceil P_1 \rceil \vee \lceil P_2 \rceil$."

III. A Gas Burner System

Our example is a simplified version of a computer controlled (on-off) gas burner described in [45]. This is a safety-critical system as an accident may occur if an excesive amount of unburned gas leaks to the environment. Small gas leaks cannot be avoided during ignition. A burning flame may also be blown out causing some gas to leak before the failure is detected. The gas burner is controlled by a thermostat and the gas is ignited by an ignition transformer, cf. Fig. 1. The informal requirements are as follows:

1) For safety, gas must never leak for more than 4 s[1] in any period of 30 s at most.
2) Heat request off shall result in the flame being off after 60 s.
3) Heat request shall after 60 s result in gas burning unless an ignite or flame failure has occurred. An ignite failure happens when gas does not ignite after 0.5 s. The flame fails if it disappears while gas is supplied.

The same timing constant has been used in 2) and 3) in order to allow a simple design.

[1] We use seconds as the time unit.

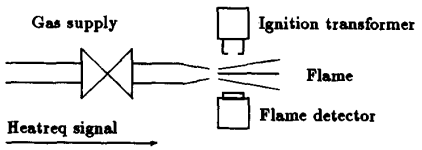


Fig. 1. Gas burner.

IV. System Model and Requirements

In order to formalize the requirements to the gas burner we introduce the following Boolean valued states:

$$Heatreq, Flame, Gas, Ignition : \textbf{Bool}$$

They express the physical state of the thermostat, the flame, the gas supply, and the ignition transformer as depicted in Fig. 1. The informal requirements for the gas burner can then be formalized using duration formulas on the system model.

1) For safety, gas must never leak for more than 4 s in any period of at most 30 s

$$Req_1 \mathrel{\widehat{=}} \ell \leq 30 \Rightarrow \int (Gas \wedge \neg Flame) \leq 4$$

This is a *critical duration* constraint.

2) Heat request off shall result in the flame being off after 60 s

$$Req_2 \mathrel{\widehat{=}} \lceil \neg Heatreq \rceil \Rightarrow (\lceil\rceil \sim 60 \rightsquigarrow \lceil \neg Flame \rceil)$$

This is a guarded progress constraint. The guard is $\lceil \neg Heatreq \rceil$ and the progress is from an initial point $\lceil\rceil$ to $\lceil \neg Flame \rceil$ within 60 s.

3) Heat request shall, after 60 s, result in gas burning, unless an ignite or flame failure has occurred

$$Req_3 \mathrel{\widehat{=}} \lceil Heatreq \rceil \Rightarrow (\lceil\rceil \sim 60 \rightsquigarrow \lceil Flame \rceil) \vee \Diamond IgniteFail \vee \Diamond FlameFail$$

An ignite failure happens when gas does not ignite within 0.5 s:

$$IgniteFail \mathrel{\widehat{=}} \neg(\lceil Gas \wedge Ignition \rceil \Rightarrow (\lceil\rceil \sim 0.5 \rightsquigarrow \lceil Flame \rceil))$$

The flame fails if it disappears while gas is supplied

$$FlameFail \mathrel{\widehat{=}} \neg(\lceil Gas \rceil \Rightarrow \neg\Diamond(\lceil Flame \rceil; \lceil \neg Flame \rceil))$$

A formula $\neg\Diamond(\lceil P \rceil; \lceil \neg P \rceil)$ means stability of P: P cannot change to $\neg P$.

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on November 27, 2009 at 04:43 from IEEE Xplore. Restrictions apply.

The total system requirements is the conjunction of the critical duration constraint Req_1 and the guarded progress constraints Req_2 and Req_3

$$ReqAll \mathrel{\widehat{=}} Req_1 \wedge Req_2 \wedge Req_3$$

The requirements should hold for any interval

$$Req \mathrel{\widehat{=}} \Box\, ReqAll$$

V. Control Model and Control Law

A *control law* expresses a top-level design of the system. It consists of a *control function* and *assumptions* about the behavior of the environment. The control function defines interaction between the control system and the environment. The assumptions define preconditions for the control law. In order to formalize the control law, the *control model* is formed. It is the system model with an additional state `P` that records the phase of operation of the controller.

A. Gas Burner Phases

We use a simple version of the control law in [45] (especially simplified with respect to error recovery) for the gas burner. It has the following *phases*:

- `Idle`: Awaits heat request; no gas and ignition. It enters the `Purge` phase on heat request.
- `Purge`: Pauses for 30 s, and then `Ignite1` is entered.
- `Ignite1`: Starts ignition and gas supply; enters the `Ignite2` phase after 1 s.
- `Ignite2`: Monitors the flame and enters the `Burn` phase if flame is sensed within 1 s.
- `Burn`: Ignition is switched off, but gas is still supplied. The `Burn` phase is stable until heat request goes off. The `Idle` phase is then entered and the gas is turned off.

We use the simple error recovery procedure of returning to `Idle`. If a flame is not sensed within 1 s in `Ignite2` (ignite failure), or if the flame disappears during the `Burn` phase (flame failure), then the `Idle` phase is entered and the gas is turned off. The 30 s `Purge` pause ensures a sufficient distance between periods with leaking gas.

B. Control Automaton

We describe the possible phase transitions by means of a finite state *automaton* with states corresponding to the phases. The automaton is shown in Fig. 2. The automaton is defined in the control model using the state `P` giving the current phase of the system.

$$\mathtt{P} : \{\mathtt{idle}, \mathtt{purge}, \mathtt{ignite1}, \mathtt{ignite2}, \mathtt{burn}\}$$

Its value domain is the finite set of phase names, and the following state assertions describe the individual phases

$$\begin{array}{rcl} \mathtt{Idle} & \mathrel{\widehat{=}} & \mathtt{P} = \mathtt{idle} \\ \mathtt{Purge} & \mathrel{\widehat{=}} & \mathtt{P} = \mathtt{purge} \\ \mathtt{Ignite1} & \mathrel{\widehat{=}} & \mathtt{P} = \mathtt{ignite1} \\ \mathtt{Ignite2} & \mathrel{\widehat{=}} & \mathtt{P} = \mathtt{ignite2} \\ \mathtt{Burn} & \mathrel{\widehat{=}} & \mathtt{P} = \mathtt{burn} \end{array}$$

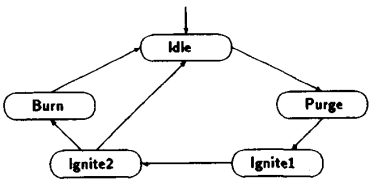


Fig. 2. Phase transitions for the gas burner.

The automaton is defined by untimed progresss constraints

$$Phases \mathrel{\widehat{=}} Init \wedge Trans$$

where $Init$ expressses that the automaton starts in the `Idle` phase

$$Init \mathrel{\widehat{=}} \lceil\,\rceil \rightarrow \lceil \mathtt{Idle} \rceil$$

and $Trans$ defines the phase transitions

$$\begin{aligned} Trans \mathrel{\widehat{=}} \Box(\ & \\ & (\lceil \mathtt{Idle} \rceil \rightarrow \lceil \mathtt{Purge} \rceil) \\ \wedge\ & (\lceil \mathtt{Purge} \rceil \rightarrow \lceil \mathtt{Ignite1} \rceil) \\ \wedge\ & (\lceil \mathtt{Ignite1} \rceil \rightarrow \lceil \mathtt{Ignite2} \rceil) \\ \wedge\ & (\lceil \mathtt{Ignite2} \rceil \rightarrow (\lceil \mathtt{Burn} \rceil \vee \lceil \mathtt{Idle} \rceil)) \\ \wedge\ & (\lceil \mathtt{Burn} \rceil \rightarrow \lceil \mathtt{Idle} \rceil)\ ) \end{aligned}$$

i.e. `Idle` is followed by `Purge`, `Purge` is followed by `Ignite1`, etc., cf. Fig. 2.

Phase Requirements. The predicate $PhaseReq$ specifies the monitoring and control of system states for each phase

$$\begin{aligned} PhaseReq \mathrel{\widehat{=}} \Box(& IdleReq \wedge PurgeReq \\ & \wedge Ignite1Req \\ & \wedge Ignite2Req \wedge BurnReq) \end{aligned}$$

In the formulas, the constant ε_1 denotes an upper bound on progress. Any move will be performed within time ε_1. The constant ε_2 gives, on the other hand, a lower bound on stability. The system will not move before time ε_2. We assume that $0 < \varepsilon_2 < \varepsilon_1$.

The `Idle` phase is stable at least ε_2 beyond $\neg Heatreq$. The `Idle` phase is left before $Heatreq$ has lasted ε_1. During the `Idle` phase Gas and $Ignition$ are turned off within ε_1

$$\begin{aligned} & IdleReq \mathrel{\widehat{=}} \\ & \quad (\lceil \neg Heatreq \rceil; \ell \leq \varepsilon_2 \Rightarrow \neg\Diamond(\lceil \mathtt{Idle} \rceil; \lceil \neg \mathtt{Idle} \rceil)) \\ & \wedge (\lceil \mathtt{Idle} \rceil \wedge \lceil Heatreq \rceil \Rightarrow \ell \leq \varepsilon_1) \\ & \wedge (\lceil \mathtt{Idle} \rceil \Rightarrow (\lceil\,\rceil \sim \varepsilon_1 \rightsquigarrow \lceil \neg Gas \wedge \neg Ignition \rceil)) \end{aligned}$$

Notice that $\ell \leq \varepsilon_1$ in the second clause can be rewritten to the progress constraint

$$\lceil\,\rceil \sim \varepsilon_1 \rightsquigarrow false$$

In conjunction with the untimed phase constraint, it will give a next phase within ε_1.

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on November 27, 2009 at 04:43 from IEEE Xplore. Restrictions apply.

A completed `Purge` phase lasts approximately 30 s, and *Gas* and *Ignition* are off within ε_1

$$\begin{aligned} &PurgeReq \mathrel{\widehat{=}} \\ &\quad (\lceil\neg\texttt{Purge}\rceil; \ell \le 30 \Rightarrow \neg\Diamond(\lceil\texttt{Purge}\rceil; \lceil\neg\texttt{Purge}\rceil)) \\ &\wedge (\lceil\texttt{Purge}\rceil \Rightarrow \ell \le 30 + \varepsilon_1) \\ &\wedge (\lceil\texttt{Purge}\rceil \Rightarrow (\lceil\rceil \sim\varepsilon_1 \rightsquigarrow \lceil\neg Gas \wedge \neg Ignition\rceil)) \end{aligned}$$

The first clause denotes an unconditional stability of `Purge` for at least 30 s.

A completed `Ignite1` phase lasts approximately 1 s, and *Gas* and *Ignition* are on within ε_1

$$\begin{aligned} &Ignite1Req \mathrel{\widehat{=}} \\ &\quad (\lceil\neg\texttt{Ignite1}\rceil; \ell \le 1 \Rightarrow \neg\Diamond(\lceil\texttt{Ignite1}\rceil; \lceil\neg\texttt{Ignite1}\rceil)) \\ &\wedge (\lceil\texttt{Ignite1}\rceil \Rightarrow \ell \le 1 + \varepsilon_1) \\ &\wedge (\lceil\texttt{Ignite1}\rceil \Rightarrow (\lceil\rceil \sim\varepsilon_1 \rightsquigarrow \lceil Gas \wedge Ignition\rceil)) \end{aligned}$$

The `Ignite2` phase does not enter `Idle` after less than 1 s and it does not enter `Burn` after less than ε_2 of *Flame*. It lasts at most $1 + \varepsilon_1$ and it is left within ε_1 if the *Flame* comes on. The `Ignite2` phase maintains *Gas* and *Ignition*

$$\begin{aligned} &Ignite2Req \mathrel{\widehat{=}} \\ &\quad (\lceil\neg\texttt{Ignite2}\rceil; \ell \le 1 \Rightarrow \neg\Diamond(\lceil\texttt{Ignite2}\rceil; \lceil\texttt{Idle}\rceil)) \\ &\wedge (\lceil\neg Flame\rceil; \ell \le \varepsilon_2 \Rightarrow \neg\Diamond(\lceil\neg\texttt{Burn}\rceil; \lceil\texttt{Burn}\rceil)) \\ &\wedge (\lceil\texttt{Ignite2}\rceil \Rightarrow \ell \le 1 + \varepsilon_1) \\ &\wedge (\lceil\texttt{Ignite2}\rceil \wedge \lceil Flame\rceil \Rightarrow \ell \le \varepsilon_1) \\ &\wedge (\lceil\texttt{Ignite2}\rceil \Rightarrow \lceil Gas \wedge Ignition\rceil) \end{aligned}$$

Note, that *Trans* implies that `Burn` can only be entered from `Ignite2`. Hence $\lceil\neg\texttt{Burn}\rceil$ in the second clause could be replaced by $\lceil\texttt{Ignite2}\rceil$.

The `Burn` phase persists ε_2 beyond *Heatreq* and *Flame*. It is left within ε_1 if *Heatreq* or *Flame* goes off. During the `Burn` phase *Gas* is maintained, but *Ignition* is switched off within ε_1

$$\begin{aligned} &BurnReq \mathrel{\widehat{=}} \\ &\quad (\lceil Heatreq \wedge Flame\rceil; \ell \le \varepsilon_2 \\ &\qquad \Rightarrow \neg\Diamond(\lceil\texttt{Burn}\rceil; \lceil\neg\texttt{Burn}\rceil)) \\ &\wedge (\lceil\texttt{Burn}\rceil \wedge \lceil\neg Heatreq \vee \neg Flame\rceil \Rightarrow \ell \le \varepsilon_1) \\ &\wedge (\lceil\texttt{Burn}\rceil \Rightarrow \lceil Gas\rceil) \\ &\wedge (\lceil\texttt{Burn}\rceil \Rightarrow (\lceil\rceil \sim\varepsilon_1 \rightsquigarrow \lceil\neg Ignition\rceil)) \end{aligned}$$

C. Assumptions

The control law also needs assumptions about the physical processes in the system. For the gas burner we have the following assumptions:

1) No gas results in no flame within 0.1 s.

$$Asm_1 \mathrel{\widehat{=}} \lceil\neg Gas\rceil \Rightarrow (\lceil\rceil \sim 0.1 \rightsquigarrow \lceil\neg Flame\rceil)$$

2) Gas does not ignite when the ignition transformer is not operating.

$$\begin{aligned} &Asm_2 \mathrel{\widehat{=}} \\ &\quad \lceil\neg Ignition\rceil \Rightarrow \neg\Diamond(\lceil\neg Flame\rceil; \lceil Flame\rceil) \end{aligned}$$

which should hold on any interval

$$Asm \mathrel{\widehat{=}} \Box(Asm_1 \wedge Asm_2)$$

VI. Correctness of Control Law

A control law is *correct* if it implements the requirements under the given assumptions. The formalized requirements, assumptions and control function allow correctness to be expressed formally as the implication

$$Asm \wedge Phases \wedge PhaseReq \Rightarrow Req$$

We have $Phases = Init \wedge Trans$, where *Init* decomposes into $\lceil\rceil$ or $\lceil\texttt{Idle}\rceil; true$. It is easy to see that requirements hold for the point interval, thus $\lceil\rceil \Rightarrow Req$. For $\lceil\texttt{Idle}\rceil; true$, we use that the automaton returns to `Idle`, such that this case is subsumed by

$$Asm \wedge Trans \wedge PhaseReq \Rightarrow Req$$

We now consider separate cases for each requirement:

1) $Asm \wedge Trans \wedge PhaseReq \Rightarrow Req_1$
2) $Asm \wedge Trans \wedge PhaseReq \Rightarrow Req_2$
3) $Asm \wedge Trans \wedge PhaseReq \Rightarrow Req_3$

A. Verification of Req_1

We first use *PhaseReq* to estimate the duration of the critical state

$$Leak \mathrel{\widehat{=}} Gas \wedge \neg Flame$$

for each phase.

We define *Limits* to be the following conjunction of estimates

$$\begin{aligned} &Limits \mathrel{\widehat{=}} \\ &\quad (\lceil\texttt{Idle}\rceil \Rightarrow \textstyle\int Gas \le \varepsilon_1) \\ &\wedge (\lceil\texttt{Purge}\rceil \Rightarrow \textstyle\int Gas \le \varepsilon_1) \\ &\wedge (\lceil\texttt{Ignite1}\rceil \Rightarrow \ell \le 1 + \varepsilon_1) \\ &\wedge (\lceil\texttt{Ignite2}\rceil \Rightarrow \ell \le 1 + \varepsilon_1) \\ &\wedge (\lceil\texttt{Burn}\rceil \Rightarrow \textstyle\int \neg Flame \le 2 \cdot \varepsilon_1) \end{aligned}$$

The deduction of

$$Asm \wedge Trans \wedge PhaseReq \Rightarrow Limits$$

can be split into simple cases for each conjunct. We take the first one in great detail

$$\begin{aligned} &IdleReq \wedge \lceil\texttt{Idle}\rceil \\ &\Rightarrow \lceil\rceil \sim\varepsilon_1 \rightsquigarrow \lceil\neg Gas \wedge \neg Ignition\rceil \\ &\Rightarrow \lceil\rceil \sim\varepsilon_1 \rightsquigarrow \lceil\neg Gas\rceil \\ &\Rightarrow \textstyle\int Gas \le \varepsilon_1 \ \vee \ (\int Gas \le \varepsilon_1; \int Gas = 0) \\ &\Rightarrow \textstyle\int Gas \le \varepsilon_1 \ \vee \ \int Gas \le \varepsilon_1 + 0 \end{aligned}$$

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on November 27, 2009 at 04:43 from IEEE Xplore. Restrictions apply.

and omit the deductions for the next three cases. The last case is

$$\begin{aligned}
&\Box\, BurnReq \wedge \Box\, Asm_2 \wedge \lceil \texttt{Burn} \rceil \\
\Rightarrow\ &\Box(\lceil \neg Flame \rceil \Rightarrow \ell \leq \varepsilon_1) \wedge \Box\, Asm_2 \\
&\wedge (\lceil\, \rceil \sim \varepsilon_1 \rightsquigarrow \lceil \neg Ignition \rceil) \\
\Rightarrow\ &\Box(\lceil \neg Flame \rceil \Rightarrow \ell \leq \varepsilon_1) \wedge \\
&(\lceil\, \rceil \sim \varepsilon_1 \rightsquigarrow \neg\Diamond(\lceil \neg Flame \rceil; \lceil Flame \rceil)) \\
\Rightarrow\ &\ell \leq \varepsilon_1 \vee (\ell \leq \varepsilon_1; \lceil Flame \rceil) \\
&\vee (\ell \leq \varepsilon_1; (\lceil \neg Flame \rceil \wedge \ell \leq \varepsilon_1)) \\
&\vee (\ell \leq \varepsilon_1; \lceil Flame \rceil \\
&\quad ; (\lceil \neg Flame \rceil \wedge \ell \leq \varepsilon_1)) \\
\Rightarrow\ &\textstyle\int \neg Flame \leq 2 \cdot \varepsilon_1
\end{aligned}$$

where we have used

$$\begin{aligned}
&\neg\Diamond(\lceil \neg Flame \rceil; \lceil Flame \rceil) \\
&\Rightarrow \lceil Flame \rceil \vee \lceil \neg Flame \rceil \vee (\lceil Flame \rceil; \lceil \neg Flame \rceil)
\end{aligned}$$

The second step in the verification is also a case analysis. The cases are defined by the initial phase of an arbitrary interval. The premise is

$$Pre_1 \mathrel{\widehat{=}} Trans \wedge \Box\, Limits \wedge \ell \leq 30$$

in

1) $Pre_1 \wedge (\lceil \texttt{Idle} \rceil; true) \Rightarrow \int Leak \leq 2 \cdot \varepsilon_1$
2) $Pre_1 \wedge (\lceil \texttt{Purge} \rceil; true) \Rightarrow \int Leak \leq 2 + 7 \cdot \varepsilon_1$
3) $Pre_1 \wedge (\lceil \texttt{Ignite1} \rceil; true) \Rightarrow \int Leak \leq 2 + 6 \cdot \varepsilon_1$
4) $Pre_1 \wedge (\lceil \texttt{Ignite2} \rceil; true) \Rightarrow \int Leak \leq 1 + 5 \cdot \varepsilon_1$
5) $Pre_1 \wedge (\lceil \texttt{Burn} \rceil; true) \Rightarrow \int Leak \leq 4 \cdot \varepsilon_1$

The calculations follow a pattern, where previous results are used

$$\begin{aligned}
&Pre_1 \wedge (\lceil \texttt{Idle} \rceil; true) \\
\Rightarrow\ &Pre_1 \wedge \\
&(\lceil \texttt{Idle} \rceil \\
&\quad \vee (\lceil \texttt{Idle} \rceil; ((\lceil \texttt{Purge} \rceil; true) \wedge \ell \leq 30))) \\
\Rightarrow\ &Limits \wedge (\lceil \texttt{Idle} \rceil \vee (\lceil \texttt{Idle} \rceil; \lceil \texttt{Purge} \rceil)) \\
\Rightarrow\ &\textstyle\int Gas \leq \varepsilon_1 \vee (\int Gas \leq \varepsilon_1; \int Gas \leq \varepsilon_1) \\
\Rightarrow\ &\textstyle\int Gas \leq 2 \cdot \varepsilon_1
\end{aligned}$$

$$\begin{aligned}
&Pre_1 \wedge (\lceil \texttt{Burn} \rceil; true) \\
\Rightarrow\ &Pre_1 \wedge (\lceil \texttt{Burn} \rceil \vee (\lceil \texttt{Burn} \rceil; \lceil \texttt{Idle} \rceil; true)) \\
\Rightarrow\ &Limits \\
&\wedge (\lceil \texttt{Burn} \rceil \vee (\lceil \texttt{Burn} \rceil; \textstyle\int Leak \leq 2 \cdot \varepsilon_1)) \\
\Rightarrow\ &\textstyle\int Leak \leq 4 \cdot \varepsilon_1
\end{aligned}$$

The cases for `Ignite1` and `Ignite2` are similar, and so is

$$\begin{aligned}
&Pre_1 \wedge (\lceil \texttt{Purge} \rceil; true) \\
\Rightarrow\ &Pre_1 \wedge (\lceil \texttt{Purge} \rceil \\
&\quad \vee (\lceil \texttt{Purge} \rceil; \lceil \texttt{Ignite1} \rceil; true)) \\
\Rightarrow\ &\textstyle\int Leak \leq 2 + 7 \cdot \varepsilon_1
\end{aligned}$$

The case analysis shows that an interval beginning in a `Purge` phase is most critical, and that Req_1 will be satisfied as long as $\varepsilon_1 \leq 2/7$.

B. Verification of Req_2

We start by proving that $\neg Heatreq$ maintains the `Idle` phase

$$\begin{aligned}
&IdleReq \wedge \lceil \neg Heatreq \rceil \wedge (\lceil \texttt{Idle} \rceil; true) \\
\Rightarrow\ &\neg\Diamond(\lceil Idle \rceil; \lceil \neg Idle \rceil) \\
&\wedge (\lceil Idle \rceil \vee \Diamond(\lceil Idle \rceil; \lceil \neg Idle \rceil)) \\
\Rightarrow\ &\lceil \texttt{Idle} \rceil
\end{aligned}$$

The deduction of Req_2 is a case analysis, depending on the initial phase of the interval. The premise Pre_2 is defined

$$\begin{aligned}
Pre_2 \mathrel{\widehat{=}}\ &\lceil \neg Heatreq \rceil \wedge \Box\, Asm_1 \wedge \Box\, Trans \\
&\wedge PhaseReq
\end{aligned}$$

and the cases are

1) $Pre_2 \wedge (\lceil \texttt{Idle} \rceil; true) \wedge \ell > 2$
   $\Rightarrow (\ell \leq 0.1 + \varepsilon_1; \lceil \neg Flame \rceil)$
2) $Pre_2 \wedge (\lceil \texttt{Burn} \rceil; true) \wedge \ell > 3$
   $\Rightarrow (\ell \leq 0.1 + 2 \cdot \varepsilon_1; \lceil \neg Flame \rceil)$
3) $Pre_2 \wedge (\lceil \texttt{Ignite2} \rceil; true) \wedge \ell > 5$
   $\Rightarrow (\ell \leq 1.1 + 3 \cdot \varepsilon_1; \lceil \neg Flame \rceil)$
4) $Pre_2 \wedge (\lceil \texttt{Ignite1} \rceil; true) \wedge \ell > 7$
   $\Rightarrow (\ell \leq 2.1 + 4 \cdot \varepsilon_1; \lceil \neg Flame \rceil)$
5) $Pre_2 \wedge (\lceil \texttt{Purge} \rceil; true) \wedge \ell > 38$
   $\Rightarrow (\ell \leq 32.1 + 5 \cdot \varepsilon_1; \lceil \neg Flame \rceil)$

The individual calculations are of the usual form:

$$\begin{aligned}
&Pre_2 \wedge (\lceil \texttt{Idle} \rceil; true) \wedge \ell > 2 \\
\Rightarrow\ &\Box(Asm_1 \wedge IdleReq) \wedge \lceil \texttt{Idle} \rceil \wedge \ell > 2 \\
\Rightarrow\ &\Box\, Asm_1 \wedge (\ell \leq \varepsilon_1; \lceil \neg Gas \rceil) \\
\Rightarrow\ &\ell \leq \varepsilon_1 + 0.1; \lceil \neg Flame \rceil
\end{aligned}$$

where we have used that $\neg Heatreq$ maintains `Idle`. The other cases are similar and the deductions are omitted.

C. Verification of Req_3

Assuming $\varepsilon_1 < 0.5$, which is compatible with $\varepsilon_1 \leq 2/7$, we deduce that `Ignite2` only leads to `Idle` in case of an $IgniteFail$.

$$\begin{aligned}
&\Box\, Ignite2Req \wedge (\lceil \texttt{Ignite1} \rceil; \lceil \texttt{Ignite2} \rceil; \lceil \texttt{Idle} \rceil) \\
\Rightarrow\ &true; (\lceil Gas \wedge Ignition \rceil \wedge \ell \geq 1 \\
&\quad \wedge \Box(\lceil Flame \rceil \Rightarrow \ell \leq 0.5)); true \\
\Rightarrow\ &\Diamond(\lceil Gas \wedge Ignition \rceil \wedge \neg(\ell \leq 0.5; \lceil Flame \rceil)) \\
\Rightarrow\ &\Diamond\, IgniteFail
\end{aligned}$$

and that a `Burn` phase is preceded by $Flame$. This is shown by contradiction

$$\begin{aligned}
&\Box\, Ignite2Req \\
&\wedge (\lceil \neg Flame \wedge \texttt{Ignite2} \rceil; \lceil \texttt{Burn} \rceil) \\
\Rightarrow\ &(\Diamond(\lceil \neg \texttt{Burn} \rceil; \lceil \texttt{Burn} \rceil) \\
&\quad \wedge \neg\Diamond(\lceil \neg \texttt{Burn} \rceil; \lceil \texttt{Burn} \rceil)); true \\
\Rightarrow\ &false
\end{aligned}$$

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on November 27, 2009 at 04:43 from IEEE Xplore. Restrictions apply.

We also deduce that Burn maintains the $Flame$ unless there is a $FlameFail$

$$\begin{aligned} & BurnReq \wedge (\lceil Flame \wedge Gas \rceil; \lceil \texttt{Burn} \rceil) \\ \Rightarrow\ & \lceil Gas \rceil \wedge (\lceil Flame \rceil; true) \\ \Rightarrow\ & \lceil Flame \rceil \\ & \quad \vee (\lceil Gas \rceil \wedge \Diamond (\lceil Flame \rceil; \lceil \neg Flame \rceil)) \\ \Rightarrow\ & \lceil Flame \rceil \vee FlameFail \end{aligned}$$

and finally that $Heatreq$ without $FlameFail$ maintains a Burn phase with $Flame$

$$\begin{aligned} & \Box(BurnReq \wedge \neg FlameFail) \wedge \lceil Heatreq \rceil \\ & \wedge (\lceil Gas \wedge Flame \rceil; \lceil \texttt{Burn} \rceil; true) \\ \Rightarrow\ & \Box(BurnReq \wedge \neg FlameFail) \wedge \lceil Heatreq \rceil \\ & \wedge ((\lceil Gas \wedge Flame \rceil; \lceil \texttt{Burn} \rceil) \\ & \quad \vee (\lceil Gas \wedge Flame \rceil; \lceil \texttt{Burn} \rceil \\ & \quad ; \lceil \neg \texttt{Burn} \rceil; true)) \\ \Rightarrow\ & (\lceil Flame \rceil; \lceil Flame \wedge \texttt{Burn} \rceil) \\ & \vee \Diamond (BurnReq \\ & \quad \wedge ((\lceil \texttt{Burn} \rceil \wedge \lceil Heatreq \wedge Flame \rceil) \\ & \quad ; \lceil \neg \texttt{Burn} \rceil)) \\ \Rightarrow\ & (\lceil Flame \rceil; \lceil Flame \wedge \texttt{Burn} \rceil) \end{aligned}$$

We can now proceed to the main deduction. This is also a case analysis, depending on the initial phase of the interval. We assume that $\varepsilon_1 \leq 1$. The premise Pre_3 is defined

$$\begin{aligned} Pre_3 \mathrel{\widehat{=}} & \lceil Heatreq \rceil \wedge Trans \wedge PhaseReq \\ & \wedge \Box \neg IgniteFail \wedge \Box \neg FlameFail \end{aligned}$$

and the cases are

1) $Pre_3 \wedge (\lceil \texttt{Ignite1} \rceil; true) \wedge \ell > 5$
   $\Rightarrow (\ell \leq 2 + 2 \cdot \varepsilon_1; \lceil Flame \rceil)$
2) $Pre_3 \wedge (\lceil \texttt{Purge} \rceil; true) \wedge \ell > 36$
   $\Rightarrow (\ell \leq 32 + 3 \cdot \varepsilon_1; \lceil Flame \rceil)$
3) $Pre_3 \wedge (\lceil \texttt{Idle} \rceil; true) \wedge \ell > 37$
   $\Rightarrow (\ell \leq 32 + 4 \cdot \varepsilon_1; \lceil Flame \rceil)$
4) $Pre_3 \wedge (\lceil \texttt{Burn} \rceil; true) \wedge \ell > 38$
   $\Rightarrow (\ell \leq 32 + 5 \cdot \varepsilon_1; \lceil Flame \rceil)$
5) $Pre_3 \wedge (\lceil \texttt{Ignite2} \rceil; true) \wedge \ell > 40$
   $\Rightarrow (\ell \leq 33 + 6 \cdot \varepsilon_1; \lceil Flame \rceil)$

The individual calculations are

$$\begin{aligned} & Pre_3 \wedge (\lceil \texttt{Ignite1} \rceil; true) \wedge \ell > 5 \\ \Rightarrow\ & Pre_3 \wedge ((\ell \leq 1 + \varepsilon_1 \wedge \lceil \texttt{Ignite1} \rceil) \\ & \quad ; (\ell \leq 1 + \varepsilon_1 \wedge \lceil \texttt{Ignite2} \rceil) \\ & \quad ; (\lceil \texttt{Burn} \rceil \vee \lceil \texttt{Idle} \rceil); true) \\ \Rightarrow\ & Pre_3 \wedge (\\ & \quad (\ell \leq 2 + 2 \cdot \varepsilon_1 \wedge (true; \lceil Flame \wedge Gas \rceil)) \\ & \quad ; \lceil \texttt{Burn} \rceil; true) \\ \Rightarrow\ & \ell \leq 2 + 2 \cdot \varepsilon_1; \lceil Flame \rceil \end{aligned}$$

where we have used the above results about Ignite2 and Burn.

The second case can, as usual, be reduced using the previous result:

$$\begin{aligned} & Pre_3 \wedge (\lceil \texttt{Purge} \rceil; true) \wedge \ell > 36 \\ \Rightarrow\ & Pre_3 \\ & \wedge (\ell \leq 30 + \varepsilon_1; ((\lceil \texttt{Ignite1} \rceil; true) \wedge \ell > 5)) \\ \Rightarrow\ & \ell \leq 32 + 3 \cdot \varepsilon_1; \lceil Flame \rceil \end{aligned}$$

and the third follows the pattern

$$\begin{aligned} & Pre_3 \wedge (\lceil \texttt{Idle} \rceil; true) \wedge \ell > 37 \\ \Rightarrow\ & \Box IdleReq \wedge \ell > 37 \\ & \wedge (\lceil \texttt{Idle} \wedge Heatreq \rceil; \ell \leq 32 + 3 \cdot \varepsilon_1 \\ & ; \lceil Flame \rceil) \\ \Rightarrow\ & \ell \leq 32 + 4 \cdot \varepsilon_1; \lceil Flame \rceil) \end{aligned}$$

The last cases have a surprise, because the observation may start just after a $FlameFail$ in the Burn phase

$$\begin{aligned} & Pre_3 \wedge (\lceil \texttt{Burn} \rceil; true) \wedge \ell > 38 \\ \Rightarrow\ & Pre_3 \wedge (\lceil \texttt{Burn} \rceil; true) \wedge \ell > 38 \\ & \wedge ((\lceil \neg Flame \rceil; true) \vee (\lceil Flame \rceil; true)) \\ \Rightarrow\ & (Pre_3 \wedge (\lceil \texttt{Burn} \rceil; true) \\ & \wedge (\lceil \neg Flame \rceil; true)) \vee \lceil Flame \rceil \\ \Rightarrow\ & (Pre_3 \wedge \ell > 38 \\ & \wedge (\ell \leq \varepsilon_1; \lceil \texttt{Idle} \rceil; true)) \vee \lceil Flame \rceil \\ \Rightarrow\ & \ell \leq 32 + 5 \cdot \varepsilon_1; \lceil Flame \rceil \end{aligned}$$

and

$$\begin{aligned} & Pre_3 \wedge (\lceil \texttt{Ignite2} \rceil; true) \wedge \ell > 40 \\ \Rightarrow\ & Pre_3 \wedge \ell > 40 \wedge (\ell \leq 1 + \varepsilon_1; \lceil \texttt{Burn} \rceil; true) \\ \Rightarrow\ & \ell \leq 33 + 6 \cdot \varepsilon_1; \lceil Flame \rceil \end{aligned}$$

In summary: the control law specifies a correct design if the constants ε_1 and ε_2 are chosen such that

$$\varepsilon_1 < 2/7 \quad \text{and} \quad 0 < \varepsilon_2 < \varepsilon_1$$

Notice that the latency ε_2 can be arbitrarily small.

The control law can almost directly be used to implement the system with a processor that accesses sensor and actuator states. The progress constraints map to assignments, whereas stability constraints map to delays, either unconditionally or in a sensor-reading loop.

There is, hovewer, an element of parallel processing in sensor readings. This indicates that a further decomposition of the design is useful.

VII. Architecture

An *architecture* specifies a collection of selected *components* together with an interconnection scheme. A component is either a subsystem or an elementary component: a program, a sensor, an actuator or a timer. A program implements a state machine that takes the system through specified phases. In the general case, state transitions may depend on values computed from data collected in previous phases. A sensor monitors a physical system state, and it is ready to communicate the

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on November 27, 2009 at 04:43 from IEEE Xplore. Restrictions apply.

TABLE II
GAS BURNER ARCHITECTURE

Component	Alphabet	Private States
Heat request sensor HS	{HeatOn. HeatOff}	$\mathtt{Ref}_{\mathtt{HS}} : \mathbb{P}\alpha\mathtt{HS}$ $Heatreq$: **Bool**
Flame sensor FS	{FlOn. FlOff}	$\mathtt{Ref}_{\mathtt{FS}} : \mathbb{P}\alpha\mathtt{FS}$ $Flame$: **Bool**
Gas actuator GA	{GasOn. GasOff}	$\mathtt{Ref}_{\mathtt{GA}} : \mathbb{P}\alpha\mathtt{GA}$ Gas : **Bool**
Ignition actuator IA	{IgnOn. IgnOff}	$\mathtt{Ref}_{\mathtt{IA}} : \mathbb{P}\alpha\mathtt{IA}$ $Ignition$: **Bool**
1 s timer T1	{Set1. Out1}	$\mathtt{Ref}_{\mathtt{T1}} : \mathbb{P}\alpha\mathtt{T1}$
30 s timer T30	{Set30. Out30}	$\mathtt{Ref}_{\mathtt{T30}} : \mathbb{P}\alpha\mathtt{T30}$
Program P	Event	$\mathtt{Ref}_{\mathtt{P}} : \mathbb{P}\alpha\mathtt{P}$

state value to a program when requested to do so. An actuator controls a physical system state, and it is ready to change the state to a communicated value from a program when so requested. A timer implements delays giving lower bounds on the duration of phases.

The interconnection scheme that we propose aims at a distributed system, where the components execute concurrently and synchronize through instantaneous, shared events. The synchronous communication paradigm is inspired by CSP [16].

The interconnection scheme uses a set of events, Event, called the system alphabet. Each component C synchronizes on a subset $\alpha\mathtt{C} \subseteq \mathtt{Event}$ called the C-component alphabet.

Each component has a designated state $\mathtt{Ref}_{\mathtt{C}}$, which records the subset of $\alpha\mathtt{C}$ that is refused at a given time. Events that are not in $\alpha\mathtt{C}$ are never refused.

A *scheduler* observes the refusal states for all components and allows the components to move whenever no one refuses an event. Whenever such a move occurs, it is recorded in a history or *trace* of events

$$\mathtt{tr} : \mathtt{Event}^*$$

It is the only shared state in the architecture and it is only *read* by components. Refusal states are private for components, and the system states are distributed among sensors and actuators.

A. Architecture of the Gas Burner

For the gas burner we use the architecture shown in Table II. In this case, where there is only one program component, system and program have the same alphabet

$$\mathtt{Event} = \alpha\mathtt{HS} \cup \alpha\mathtt{FS} \cup \alpha\mathtt{GA} \cup \alpha\mathtt{IA} \cup \alpha\mathtt{T1} \cup \alpha\mathtt{T30}$$

The events have the informal meaning: heat request on, heat request off, ..., set 30 s timer, expiration of 30 s timer.

We let e range over Event and s over $\mathtt{Event}^*$ in the following, and we denote the projection of tr on a component alphabet $\alpha\mathtt{C}$ by $\mathtt{tr}_{\mathtt{C}}$

$$\mathtt{tr}_{\mathtt{C}} \mathrel{\widehat{=}} \mathtt{tr} \upharpoonright \alpha\mathtt{C}$$

B. General Properties

Trace tr accumulates the history of events for the system, i.e., an event occurring at time t is appended to the previous value of tr. The trace is empty initially

$$Inittr \mathrel{\widehat{=}} \lceil\ \rceil \rightarrow \lceil \mathtt{tr} = \langle\rangle \rceil ; true$$

and tr is an *increasing* function of time (in the prefix ordering of sequences of events)

$$TrIncr \mathrel{\widehat{=}} \lceil \mathtt{tr} = \mathtt{s}_1 \rceil ; \lceil \mathtt{tr} = \mathtt{s}_2 \rceil \Rightarrow \lceil \mathtt{s}_1 \preceq \mathtt{s}_2 \rceil$$

This constraint allows a finite set of events to happen at one time.

Event e is not appended to trace if it is refused by any component C

$$WfRef \mathrel{\widehat{=}} \lceil \bigvee_{\mathtt{C} \in \mathtt{Comp}} \mathtt{e} \in \mathtt{Ref}_{\mathtt{C}} \rceil \Rightarrow \mathrm{stable}\,(\mathtt{tr} \upharpoonright \{\mathtt{e}\})$$

where Comp is the finite set of components and where

$$\mathrm{stable}\,(\mathtt{tr} \upharpoonright \mathtt{A}) \mathrel{\widehat{=}} \exists \mathtt{s} : \mathtt{A}^* \bullet \lceil \mathtt{tr} \upharpoonright \mathtt{A} = \mathtt{s} \rceil$$

expresses stability for the projection $\mathtt{tr} \upharpoonright \mathtt{A}$ of the trace tr on a subset $\mathtt{A} \subseteq \mathtt{Event}$.

Constraints $WfRef$ and $TrIncr$ allow a finite set of events to occur instantaneously and does not force anything to happen. The actual occurrence of events is controlled by a *Scheduler*.

The general properties of trace and refusals are collected in the formula

$$General \mathrel{\widehat{=}} Inittr \wedge \Box (TrIncr \wedge WfRef)$$

C. Scheduler

The scheduler is the synchronization agent for the distributed system

$$Scheduler \mathrel{\widehat{=}} \Box\ (TrOne \wedge TrStep \wedge Progress)$$

Progress is ensured by insisting that the the trace tr remains stable at most δ_1 when some event e is not refused by all components

$$Progress \mathrel{\widehat{=}} \quad \mathrm{stable}\,(\mathtt{tr}) \wedge \lceil \bigwedge_{\mathtt{C} \in \mathtt{Comp}} \mathtt{e} \notin \mathtt{Ref}_{\mathtt{C}} \rceil \Rightarrow \ell < \delta_1$$

It follows that if an event e is accepted by all components for time δ_1 then the trace becomes extended with some event (which may be different from e).

We shall not use true concurrency,[2] so we introduce special constraints that force events to happen one at a time and with a minimal distance δ_2

$$TrOne \mathrel{\widehat{=}} \lceil \mathtt{tr} = \mathtt{s}_1 \rceil ; \lceil \mathtt{tr} = \mathtt{s}_2 \rceil \Rightarrow \lceil \#\mathtt{s}_2 \leq \#\mathtt{s}_1 + 1 \rceil$$

[2] If we allow true concurrency, the refusal shall distinguish between e.g., willingness to participate in each of two events and willingness to participate in both events simultaneously. This distinction is possible with a more complicated definition of the refusal.

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on November 27, 2009 at 04:43 from IEEE Xplore. Restrictions apply.

and

$$TrStep \mathrel{\widehat{=}} \lceil \mathtt{tr} \neq \mathtt{s}_1 \rceil ; \lceil \mathtt{tr} = \mathtt{s}_1 \rceil ; \lceil \mathtt{tr} \neq \mathtt{s}_1 \rceil \Rightarrow \ell > \delta_2$$

where # denotes the length of a sequence. Consistency dictates, that $\delta_2 < \delta_1$.

D. Program

In order to specify the program for the gas burner we introduce the following regular expression over the alphabet Event

$$\begin{aligned} \mathtt{cycle} = {} & \mathtt{IgnOff.GasOff.HeatOn.Set30.} \\ & \mathtt{Out30.Set1.IgnOn.GasOn.} \\ & \mathtt{Out1.Set1.} \\ & (\mathtt{FlOn.IgnOff.(HeatOff} + \mathtt{FlOff}) \\ & + \mathtt{Out1}) \\ \mathtt{ptrace} = {} & \mathtt{pref(cycle^*)} \end{aligned}$$

where "pref" denotes the prefix closure operation on a regular expression.

The program is specified by a refusal constraint: the program refuses an event e if and only if the event does not extend the program trace $\mathtt{tr}_P$ (= tr) to a member of ptrace

$$Program \mathrel{\widehat{=}} \lceil \mathtt{e} \in \mathtt{Ref}_P \rceil \Leftrightarrow \lceil \mathtt{tr}^\frown \langle \mathtt{e} \rangle \notin \mathtt{ptrace} \rceil$$

It can be verified that trace tr always belongs to ptrace

$$Scheduler \wedge General \wedge Program \Rightarrow PTrace$$

where

$$Ptrace \mathrel{\widehat{=}} \Box(\lceil \rceil \vee \lceil \mathtt{tr} \in \mathtt{ptrace} \rceil)$$

In order to express the phase in terms of the trace, we introduce the projection $\mathtt{tr}_{In} = \mathtt{tr} \restriction \mathtt{In}$ of trace tr onto the subalphabet

$$\mathtt{In} \mathrel{\widehat{=}} \{\mathtt{FlOn}, \mathtt{FlOff}, \mathtt{HeatOn}, \mathtt{HeatOff}, \mathtt{Out1}, \mathtt{Out30}\}$$

It is obvious that $\mathtt{tr}_{In}$ will have the projected property of $Ptrace$

$$PInTrace \mathrel{\widehat{=}} \Box\,(\lceil \rceil \vee \lceil \mathtt{tr}_{In} \in \mathtt{inptrace} \rceil)$$

where the regular expressions

$$\begin{aligned} \mathtt{incycle} = {} & \mathtt{HeatOn.Out30.Out1.} \\ & (\mathtt{FlOn.(HeatOff} + \mathtt{FlOff}) + \mathtt{Out1}) \\ \mathtt{inptrace} = {} & \mathtt{pref(incycle^*)} \end{aligned}$$

are the projections of cycle and ptrace on In.

The phases are now described by

$$\begin{aligned} \mathtt{Idle} &\Leftrightarrow \mathtt{tr}_{In} \in \mathtt{incycle}^* \\ \mathtt{Purge} &\Leftrightarrow \mathtt{tr}_{In} \in \mathtt{incycle}^*.\mathtt{HeatOn} \\ \mathtt{Ignite1} &\Leftrightarrow \mathtt{tr}_{In} \in \mathtt{incycle}^*.\mathtt{HeatOn.Out30} \\ \mathtt{Ignite2} &\Leftrightarrow \mathtt{tr}_{In} \in \mathtt{incycle}^*.\mathtt{HeatOn.Out30.Out1} \\ \mathtt{Burn} &\Leftrightarrow \mathtt{tr}_{In} \in \mathtt{incycle}^*.\mathtt{HeatOn.Out30.Out1.FlOn} \end{aligned}$$

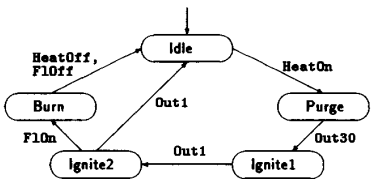


Fig. 3. Phase transitions and in-events.

It follows that the phase changes correspond to occurences of the in-event as shown in Fig. 3, i.e.,

$$General \wedge Scheduler \wedge Program \Rightarrow Phases$$

E. Sensors

The heat request sensor (HS) is specified as a conjunction

$$HeatReqSensor \mathrel{\widehat{=}} \Box(HSRef \wedge HSReady)$$

of a stability constraint: The HeatOn event is refused if $Heatreq$ has been on for less than time δ_2 and the HeatOff event is refused if $Heatreq$ has been off for less than time δ_2

$$\begin{aligned} HSRef \mathrel{\widehat{=}} {} & \\ & (\lceil \neg Heatreq \rceil ; \ell \leq \delta_2 \Rightarrow \lceil \mathtt{HeatOn} \in \mathtt{Ref}_{HS} \rceil) \\ \wedge\, & (\lceil Heatreq \rceil ; \ell \leq \delta_2 \Rightarrow \lceil \mathtt{HeatOff} \in \mathtt{Ref}_{HS} \rceil) \end{aligned}$$

and a progress constraint: The HeatOn event is not refused when has $Heatreq$ been on for time δ_1 and the HeatOff event is not refused when $Heatreq$ has been off for time δ_1

$$\begin{aligned} HSReady \mathrel{\widehat{=}} {} & \\ & (\lceil Heatreq \rceil \Rightarrow (\lceil \rceil \sim \delta_1 \rightsquigarrow \lceil \mathtt{HeatOn} \notin \mathtt{Ref}_{HS} \rceil)) \\ \wedge\, & (\lceil \neg Heatreq \rceil \Rightarrow (\lceil \rceil \sim \delta_1 \rightsquigarrow \lceil \mathtt{HeatOff} \notin \mathtt{Ref}_{HS} \rceil)) \end{aligned}$$

Flame sensor (FS) has a similar specification

$$FlameSensor \mathrel{\widehat{=}} \Box(FSRef \wedge FSReady)$$

where

$$\begin{aligned} FSRef \mathrel{\widehat{=}} {} & \\ & (\lceil \neg Flame \rceil ; \ell \leq \delta_2 \Rightarrow \lceil \mathtt{FlOn} \in \mathtt{Ref}_{FS} \rceil) \\ \wedge\, & (\lceil Flame \rceil ; \ell \leq \delta_2 \Rightarrow \lceil \mathtt{FlOff} \in \mathtt{Ref}_{FS} \rceil) \end{aligned}$$

and

$$\begin{aligned} FSReady \mathrel{\widehat{=}} {} & \\ & (\lceil Flame \rceil \Rightarrow (\lceil \rceil \sim \delta_1 \rightsquigarrow (\lceil \mathtt{FlOn} \notin \mathtt{Ref}_{FS} \rceil)) \\ \wedge\, & (\lceil \neg Flame \rceil \Rightarrow (\lceil \rceil \sim \delta_1 \rightsquigarrow (\lceil \mathtt{FlOff} \notin \mathtt{Ref}_{FS} \rceil)) \end{aligned}$$

F. Actuators

The gas actuator (GA) is specified as a conjunction

$$GasActuator \mathrel{\widehat{=}} \Box(GAReady \wedge GAAct)$$

of two progress constraints: the gas actuator never refuses GasOn or GasOff events

$$GAReady \mathrel{\widehat{=}} \lceil \mathtt{Ref}_{GA} = \{\} \rceil \vee \lceil \rceil$$

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on November 27, 2009 at 04:43 from IEEE Xplore. Restrictions apply.

and, for an empty trace, the gas is off. Otherwise, the value stabilizes δ_1 after the latest GasOn or GasOff event

$$\begin{aligned} GAAct \mathrel{\widehat{=}} & \\ & (\lceil \mathtt{tr}_{\mathtt{GA}} = \langle\rangle \rceil \Rightarrow \lceil \neg Gas \rceil) \\ \wedge\ & (\lceil \mathrm{last}(\mathtt{tr}_{\mathtt{GA}}) = \mathtt{GasOff} \rceil \\ & \quad \Rightarrow (\lceil\ \rceil \sim \delta_1 \rightsquigarrow \lceil \neg Gas \rceil)) \\ \wedge\ & (\lceil \mathrm{last}(\mathtt{tr}_{\mathtt{GA}}) = \mathtt{GasOn} \rceil \\ & \quad \Rightarrow (\lceil\ \rceil \sim \delta_1 \rightsquigarrow \lceil Gas \rceil)) \end{aligned}$$

where last(s) denotes the last element in the (nonempty) sequence s.

The ignition actuator (IA) has a similar specification

$$IgnitionActuator \mathrel{\widehat{=}} \Box(IAReady \wedge IAAct)$$

where

$$IAReady \mathrel{\widehat{=}} \lceil \mathtt{Ref}_{\mathtt{IA}} = \{\} \rceil \vee \lceil\ \rceil$$

and

$$\begin{aligned} IAAct \mathrel{\widehat{=}} & \\ & (\lceil \mathtt{tr}_{\mathtt{IA}} = \langle\rangle \rceil \\ & \quad \Rightarrow \lceil \neg Ignition \rceil) \\ \wedge\ & (\lceil \mathrm{last}(\mathtt{tr}_{\mathtt{IA}}) = \mathtt{IgnOff} \rceil \\ & \quad \Rightarrow (\lceil\ \rceil \sim \delta_1 \rightsquigarrow \lceil \neg Ignition \rceil)) \\ \wedge\ & (\lceil \mathrm{last}(\mathtt{tr}_{\mathtt{IA}}) = \mathtt{IgnOn} \rceil \\ & \quad \Rightarrow (\lceil\ \rceil \sim \delta_1 \rightsquigarrow \lceil Ignition \rceil)) \end{aligned}$$

G. Timers

The 1-s timer (T1) is specified as a conjunction

$$Timer1 \mathrel{\widehat{=}} \Box(T1Ready \wedge T1Ref)$$

of two progress constraints: the timer never refuses the set-timer event Set1, and accepts the time-out event Out1 at most 1 s after the last set-timer event.

$$\begin{aligned} T1Ready \mathrel{\widehat{=}} & \\ & (\lceil\ \rceil \vee \lceil \mathtt{Set1} \notin \mathtt{Ref}_{\mathtt{T1}} \rceil) \\ \wedge\ & \lceil \mathrm{last}(\mathtt{tr}_{\mathtt{T1}}) = \mathtt{Set1} \rceil \wedge \mathrm{stable}(\mathtt{tr}_{\mathtt{T1}}) \\ & \quad \Rightarrow (\lceil\ \rceil \sim 1 \rightsquigarrow \lceil \mathtt{Out1} \notin \mathtt{Ref}_{\mathtt{T1}} \rceil) \end{aligned}$$

and a refusal constraint: The time-out event Out1 is only accepted 1 s after a set timer event Set1.

$$\begin{aligned} & T1Ref \mathrel{\widehat{=}} \\ & \lceil \mathtt{tr}_{\mathtt{T1}} = \mathtt{s} \rceil ; \lceil \mathtt{tr}_{\mathtt{T1}} = \mathtt{s}^\frown\langle \mathtt{Set1} \rangle \rceil ; \lceil \mathtt{Out1} \notin \mathtt{Ref}_{\mathtt{T1}} \rceil \\ & \Rightarrow \ell > 1 \end{aligned}$$

From the timer specification it is easy to deduce

$$\begin{aligned} & \lceil \mathtt{tr}_{\mathtt{T1}} = \mathtt{s} \rceil ; \lceil \mathtt{tr}_{\mathtt{T1}} = \mathtt{s}^\frown\langle \mathtt{Set1} \rangle \rceil \\ & \quad ; \lceil \mathtt{tr}_{\mathtt{T1}} = \mathtt{s}^\frown\langle \mathtt{Set1}, \mathtt{Out1} \rangle \rceil \\ & \Rightarrow \ell > 1 \end{aligned}$$

The other timer has a similar specification

$$Timer30 \mathrel{\widehat{=}} \Box(T30Ready \wedge T30Ref)$$

where

$$\begin{aligned} & T30Ready \mathrel{\widehat{=}} \\ & (\lceil\ \rceil \vee \lceil \mathtt{Set30} \notin \mathtt{Ref}_{\mathtt{T30}} \rceil) \\ & \wedge \lceil \mathrm{last}(\mathtt{tr}_{\mathtt{T30}}) = \mathtt{Set30} \rceil \wedge \mathrm{stable}(\mathtt{tr}_{\mathtt{T30}}) \\ & \quad \Rightarrow (\lceil\ \rceil \sim 30 \rightsquigarrow \lceil \mathtt{Out30} \notin \mathtt{Ref}_{\mathtt{T30}} \rceil) \end{aligned}$$

and

$$\begin{aligned} & T30Ref \mathrel{\widehat{=}} \\ & \lceil \mathtt{tr}_{\mathtt{T30}} = \mathtt{s} \rceil ; \lceil \mathtt{tr}_{\mathtt{T30}} = \mathtt{s}^\frown\langle \mathtt{Set30} \rangle \rceil \\ & \quad ; \lceil \mathtt{Out30} \notin \mathtt{Ref}_{\mathtt{T30}} \rceil \\ & \quad \Rightarrow \ell > 30 \end{aligned}$$

H. Composition

The *composite* system is specified by the conjunction of the component specifications, the scheduler specification and the general properties

$$\begin{aligned} System \mathrel{\widehat{=}} & \\ & Program \wedge HeatReqSensor \wedge FlameSensor \\ & \wedge GasActuator \wedge IgnitionActuator \wedge Timer1 \\ & \wedge Timer30 \wedge Scheduler \wedge General \end{aligned}$$

VIII. Correctness of Architecture

We have already argued that the program implements the automaton

$$General \wedge Scheduler \wedge Program \Rightarrow Phases$$

It remains to verify that the architecture refines the Phase requirements.

Here we use that *Program* is the only component that delays output events, i.e., events in the set

$$\mathtt{Out} \mathrel{\widehat{=}} \mathtt{Event} \backslash \mathtt{In}$$

The scheduler ensures that the delay is, at most, δ_1. Thus we have

$$System \Rightarrow Outprogress$$

where

$$Outprogress \mathrel{\widehat{=}} \Box(\lceil \mathtt{tr} = \mathtt{s} \wedge \mathrm{nextOut}\ (\mathtt{s}) \rceil \Rightarrow \ell < \delta_1)$$

and

$$\mathrm{nextOut}(\mathtt{s}) \mathrel{\widehat{=}} \exists \mathtt{e} : \mathtt{Out} \bullet \mathtt{s}^\frown\langle \mathtt{e} \rangle \in \mathtt{ptrace}$$

From *Outprogress* it follows that a sequence of n out-events will take at most $n \cdot \delta_1$ time units to complete.

Each phase is defined by a sequence of out-events and is terminated by a single in-event or a choice between in-events, cf. the definition of ptrace and inptrace. The transition to a next phase will only happen when both the program and the sensor or timer are ready.

Using these facts about the trace and the results about *Outprogress* we have the following:

$$\begin{aligned} & System \Rightarrow \\ & \Box(LIdle \wedge LPurge \wedge LIgnite1 \wedge LIgnite2 \wedge LBurn) \end{aligned}$$

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on November 27, 2009 at 04:43 from IEEE Xplore. Restrictions apply.

where the lemmas for the phases are

$$
\begin{aligned}
&LIdle \mathrel{\widehat{=}} \\
&(\lceil \texttt{Ignite1} \rceil \Rightarrow (\lceil \rceil \sim 3 \cdot \delta_1 \rightsquigarrow \\
&\quad (\text{stable}(\texttt{tr}) \wedge \lceil \text{last}_2(\texttt{tr}) = \langle \texttt{IgnOff}, \texttt{GasOff} \rangle \rceil \\
&\quad \wedge \lceil \texttt{HeatOn} \notin \texttt{Ref}_{\texttt{P}} \rceil))) \\
&\wedge (\lceil \texttt{Idle} \rceil ; \lceil \neg \texttt{Idle} \rceil \Rightarrow \Diamond \lceil \texttt{HeatOn} \notin \texttt{Ref}_{\texttt{HS}} \wedge \texttt{Idle} \rceil)
\end{aligned}
$$

$$
\begin{aligned}
&LPurge \mathrel{\widehat{=}} \\
&(\lceil \texttt{Purge} \rceil \Rightarrow \lceil \text{last}_2(\texttt{tr}_{\texttt{GA} \cup \texttt{IA}}) = \langle \texttt{IgnOff}, \texttt{GasOff} \rangle \rceil) \\
&\wedge (\lceil \texttt{Purge} \rceil \Rightarrow (\lceil \rceil \sim 2 \cdot \delta_1 \rightsquigarrow \\
&\quad (\text{stable}(\texttt{tr}) \wedge \lceil \text{last}(\texttt{tr}) = \texttt{Set30} \rceil \\
&\quad \wedge \lceil \texttt{Out30} \notin \texttt{Ref}_{\texttt{P}} \rceil))) \\
&\wedge (\lceil \texttt{Purge} \rceil ; \lceil \neg \texttt{Purge} \rceil \\
&\quad \Rightarrow \Diamond \lceil \texttt{Out30} \notin \texttt{Ref}_{\texttt{T30}} \wedge \texttt{Purge} \rceil)
\end{aligned}
$$

$$
\begin{aligned}
&LIgnite1 \mathrel{\widehat{=}} \\
&(\lceil \texttt{Idle} \rceil \Rightarrow (\lceil \rceil \sim 4 \cdot \delta_1 \rightsquigarrow \\
&\quad (\text{stable}(\texttt{tr}) \wedge \lceil \text{last}_3(\texttt{tr}) = \langle \texttt{Set1}, \texttt{IgnOn}, \texttt{GasOn} \rangle \rceil \\
&\quad \wedge \lceil \texttt{Out1} \notin \texttt{Ref}_{\texttt{P}} \rceil))) \\
&\wedge (\lceil \texttt{Ignite1} \rceil ; \lceil \neg \texttt{Ignite1} \rceil \\
&\quad \Rightarrow \Diamond \lceil \texttt{Out1} \notin \texttt{Ref}_{\texttt{T1}} \wedge \texttt{Ignite1} \rceil)
\end{aligned}
$$

$$
\begin{aligned}
&LIgnite2 \mathrel{\widehat{=}} \\
&(\lceil \texttt{Ignite2} \rceil \Rightarrow \lceil \text{last}_2(\texttt{tr}_{\texttt{GA} \cup \texttt{IA}}) = \langle \texttt{IgnOn}, \texttt{GasOn} \rangle \rceil) \\
&(\lceil \texttt{Ignite2} \rceil \Rightarrow (\lceil \rceil \sim 3 \cdot \delta_1 \rightsquigarrow \\
&\quad (\text{stable}(\texttt{tr}) \wedge \lceil \text{last}(\texttt{tr}) = \texttt{Set1} \rceil \\
&\quad \wedge \lceil \texttt{FlOn} \notin \texttt{Ref}_{\texttt{P}} \wedge \texttt{Out1} \notin \texttt{Ref}_{\texttt{P}} \rceil))) \\
&\wedge (\lceil \texttt{Ignite2} \rceil ; \lceil \texttt{Idle} \rceil \\
&\qquad \Rightarrow \Diamond \lceil \texttt{Out1} \notin \texttt{Ref}_{\texttt{T1}} \wedge \texttt{Ignite2} \rceil) \\
&\wedge (\lceil \texttt{Ignite2} \rceil ; \lceil \texttt{Burn} \rceil \\
&\qquad \Rightarrow \Diamond \lceil \texttt{FlOn} \notin \texttt{Ref}_{\texttt{FS}} \wedge \texttt{Ignite2} \rceil)
\end{aligned}
$$

$$
\begin{aligned}
&LBurn \mathrel{\widehat{=}} \\
&(\lceil \texttt{Burn} \rceil \Rightarrow \lceil \text{last}(\texttt{tr}_{\texttt{GA}}) = \texttt{GasOn} \rceil) \\
&\wedge (\lceil \texttt{Burn} \rceil \Rightarrow (\lceil \rceil \sim 2 \cdot \delta_1 \rightsquigarrow \\
&\quad (\text{stable}(\texttt{tr}) \wedge \lceil \text{last}(\texttt{tr}) = \texttt{IgnOff} \rceil \\
&\quad \wedge \lceil \texttt{FlOff} \notin \texttt{Ref}_{\texttt{P}} \wedge \texttt{HeatOff} \notin \texttt{Ref}_{\texttt{P}} \rceil))) \\
&\wedge (\lceil \texttt{Burn} \rceil ; \lceil \neg \texttt{Burn} \rceil \Rightarrow \\
&\Diamond \lceil (\texttt{FlOff} \notin \texttt{Ref}_{\texttt{FS}} \vee \texttt{HeatOff} \notin \texttt{Ref}_{\texttt{HS}}) \\
&\qquad \wedge \texttt{Burn} \rceil)
\end{aligned}
$$

We have used $\text{last}_n(\texttt{s})$, $n \geq 1$ to denote the subsequence formed by the last n elements of s ($\text{last}_n(\texttt{s})$ s only defined when $\#\texttt{s} \geq n$).

We can now illustrate the verification of phase contraints by

$$System \Rightarrow IdleReq.$$

Stability under $\neg Heatreq$ is verified by deduction to a contradiction. We assume that $\varepsilon_2 \leq \delta_2$

$$
\begin{aligned}
&HSRef \wedge \Box LIdle \wedge (\lceil \neg Heatreq \rceil ; \ell \leq \varepsilon_2) \\
&\quad \wedge \Diamond (\lceil \texttt{Idle} \rceil ; \lceil \neg \texttt{Idle} \rceil) \\
&\Rightarrow \Box LIdle \wedge \lceil \texttt{HeatOn} \in \texttt{Ref}_{\texttt{HS}} \rceil \\
&\quad \wedge \Diamond (\lceil \texttt{Idle} \rceil ; \lceil \neg \texttt{Idle} \rceil \\
&\Rightarrow \Diamond (\lceil \texttt{HeatOn} \in \texttt{Ref}_{\texttt{HS}} \rceil \wedge \lceil \texttt{HeatOn} \notin \texttt{Ref}_{\texttt{HS}} \rceil) \\
&\Rightarrow false.
\end{aligned}
$$

Progress to `Purge` is shown by a similar deduction where we assume that $4 \cdot \delta_1 < \varepsilon_1$

$$
\begin{aligned}
&Scheduler \wedge \Box LIdle \wedge HSReady \\
&\quad \wedge \lceil \texttt{Idle} \rceil \wedge \lceil Heatreq \rceil \wedge \ell > \varepsilon_1 \\
&\Rightarrow Scheduler \wedge \\
&\quad (\lceil \rceil \sim 3 \cdot \delta_1 \rightsquigarrow (\lceil \texttt{HeatOn} \notin \texttt{Ref}_{\texttt{P}} \rceil \wedge \text{stable}(\texttt{tr})) \\
&\quad \wedge (\lceil \rceil \sim \delta_1 \rightsquigarrow \lceil \texttt{HeatOn} \notin \texttt{Ref}_{\texttt{HS}} \rceil) \wedge \ell > \varepsilon_1 \\
&\Rightarrow false.
\end{aligned}
$$

Progress to $\neg Gas$ and $\neg Ignition$ follows from

$$
\begin{aligned}
&\Box LIdle \wedge GAAct \wedge IAAct \wedge \lceil \texttt{Idle} \rceil \\
&\Rightarrow (\lceil \rceil \sim 4 \cdot \varepsilon_1 \rightsquigarrow \lceil \neg Gas \rceil) \\
&\quad \wedge (\lceil \rceil \sim 4 \cdot \varepsilon_1 \rightsquigarrow \lceil \neg Ignition \rceil)
\end{aligned}
$$

Stability of `Purge` is verified as follows:

$$
\begin{aligned}
&\Box T30Ref \wedge \Box LPurge \wedge (\lceil \neg \texttt{Purge} \rceil ; \ell \leq 30) \\
&\quad \wedge \Diamond (\lceil \texttt{Purge} \rceil ; \lceil \neg \texttt{Purge} \rceil) \\
&\Rightarrow \Diamond (\ell \leq 30 \\
&\qquad \wedge (\lceil \texttt{tr} = \texttt{s} \rceil ; \lceil \texttt{tr} = \texttt{s}^\frown \langle \texttt{Set30} \rangle \rceil \\
&\qquad ; \lceil \texttt{Out30} \notin \texttt{Ref}_{\texttt{T30}} \rceil) \\
&\qquad \wedge T30Ref) \\
&\Rightarrow false.
\end{aligned}
$$

Verification of progress in `Ignite1` requires $3 \cdot \delta_1 < \varepsilon_1$.

For `Ignite2` we have a new kind of deduction in order to verify $\lceil Gas \wedge Ignition \rceil$ throughout the phase. We look for a contradiction of

$$\Diamond \lceil \neg (Gas \wedge Ignition) \wedge Ignite2 \rceil$$

First, we have, from the properties of `ptrace`,

$$
\begin{aligned}
&System \wedge (true ; \lceil \texttt{Ignite2} \rceil) \\
&\quad \Rightarrow true ; \lceil \texttt{Purge} \rceil ; \lceil \texttt{Ignite1} \rceil ; \lceil \texttt{Ignite2} \rceil
\end{aligned}
$$

We can now deduce

$$
\begin{aligned}
&System \wedge \Diamond \lceil \neg (Gas \wedge Ignition) \wedge \texttt{Ignite2} \rceil \\
&\Rightarrow \Box (LIgnite1 \wedge LIgnite2 \wedge GAAct \wedge AAAct) \\
&\quad \wedge \Diamond (\lceil \texttt{Purge} \rceil ; \lceil \texttt{Ignite1} \rceil \\
&\qquad ; (\lceil \texttt{Ignite2} \rceil \wedge \Diamond \lceil \neg (Gas \wedge Ignition) \rceil)) \\
&\Rightarrow \Box (GAAct \wedge IAAct) \wedge \\
&\quad \Diamond (\ell \geq 1 - 3 \cdot \delta_1 \\
&\qquad \wedge \lceil \text{last}_2 (\texttt{tr}_{\texttt{GA} \cup \texttt{IA}}) = \langle \texttt{IgnOn}, \texttt{GasOn} \rangle \rceil \\
&\qquad \wedge (true ; \lceil \neg (Gas \wedge Ignition) \rceil)) \\
&\Rightarrow false.
\end{aligned}
$$

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on November 27, 2009 at 04:43 from IEEE Xplore. Restrictions apply.

provided that $\delta_1 \leq 1 - 3 \cdot \delta_1$; i.e., $4 \cdot \delta_1 \leq 1$ or $\delta_1 < 1/4$.

Maintenance of Gas in the `Burn` phase requires a similar argument.

In summary: whenever δ_1 is chosen such that $5 \cdot \delta_1 < \varepsilon_1 < 2/7$ the distributed system will work. Notice that the minimal latency δ_2 can be arbitrarily small because ε_2 has no lower bound. In other words: a faster computer will not invalidate the architecture. This is achieved by using explicit timers to implement delays.

IX. Conclusion

We have illustrated an approach to requirements engineering and design of real-time systems using mathematical specifications of system requirements and system design. We have demonstrated how mathematical reasoning is used in verifying that designs satisfy requirements and in proving that a more detailed distributed design satisfies an abstract centralized design.

Duration calculus has been our tool. This logic combines central properties of integral calculus with the serial composition ability of interval logic and the parallel composition properties of usual logic. Furthermore, it has a conventional dynamic system as model, which gives a strong link to well established mathematical theories used in control engineering.

The approach is still developing as we gain more understanding of the abilities of duration calculus, and as we see how good design principles are reflected in the formulas. We now discuss some issues in the three stages of the approach.

A. System Model and Requirements

As presented here, the requirements are straightforward formalizations of user expectations. It would be useful to have a more systematic approach for eliciting these expectations. One possibility that we would like to investigate is to derive the safety requirements from the results of a systematic safety analysis of the system. This could, for instance, be done by formalizing the results of fault tree analysis, cf. the approach in the British Ministry of Defence draft standard for software in safety critical systems [27], [28] or the approach suggested in [39].

B. Control Model and Law

The use of a finite state machine to give the control structure is not new, but we see it as a strength that the paradigm is used by other researchers (see the introduction). Our main reason for chosing such a restricted structure is that we would like a design to be consistent. It is fairly obvious that the specification of the automaton $Phases$ has a model. We have taken some pains to give the phase-state constraints $PhaseReq$ a form that ensures consistency. The system states $HeatReq$ and $Flame$ occurs only in the premise of the individual phase requirements, and only in the form of single occurrences of mutually exclusive state assertions, e.g. $\neg HeatReq$ and $Heatreq$. These states are thus free to vary in a model. The controlled states $Ignition$ and Gas occur in the same form, but in the consequence for the individual phase requirements. It is thus posible to assign consistent values for each phase. We can also check that upper bounds on the duration of a single phase are higher than the possible lower bounds. For example, for `Ignite2` we have an upper bound of $1 + \varepsilon_1$ and lower bounds of 0 or 1. Thus the $Phases$ and $PhaseReq$ constraints have a reasonable model. The assumptions Asm might spoil this, but we have used a form where premisses contain mutually disjoint assertions on the controlled states, and consequences are constraints on the free state $Flame$. This leaves $Heatreq$ unconstrained, as we expect it to be.

We are less concerned with consistency of top-level requirements. In fact, we expect them to be inconsistent at the start of a development—user expectations are generally too high. They will be relaxed during the design verification activity because it would be impossible to find a consistent design that satisfies all of them.

The control law specification could be used directly for development of a program with shared variables, linking to such approaches as [6], [8], and [24]. We might add that a control law can be refined by expanding a phase into subphases, e.g., the division of the `Ignite` phase. We do not foresee problems with such serial refinements. They are analogous to refinement of sequential programs.

The present formulation of control laws would also aid in a generalization to hybrid systems [25], [26], and [30] , where continuous physical states can be constrained by differential equations in the individual phases or in the assumptions. This requires an extension of the duration concept to properties of continuous states and a notation for initial values in an interval for such states. Such an extension was introduced in [37].

C. Verification

With the restricted form of predicates used in the verification, there is some indication that these might be decidable. This would allow mechanical support for the tedious parts of the calculations. If a finite state machine is inadequate or inconvenient, it is possible to add further control state variables and thus get the full power of a Turing machine. This will, however, make verification more difficult because the phases may be interrelated in nonobvious ways. It would also make mechanized verification support difficult.

D. Architecture, Components, and Scheduling

We have not pursued the state machine approach for a distributed architecture because of the involved proof obligations for shared variables. We have also refrained from pursuing an asynchronous event approach because timing constraints would have to be formulated as constraints on arrival and departure time for elements in unbounded buffers. We have seen dynamic buffer systems in practice, and they have not convinced us that arguments for timeliness are assisted by having global buffer pools. In the presentation, we have tried to build on the fundamental work by Reed on Timed CSP [38]. We have, however, yet to fully clearify the relations between our model and his hierarchy. The concept of a scheduler as an explicit component is elaborated in [48].

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on November 27, 2009 at 04:43 from IEEE Xplore. Restrictions apply.

APPENDIX
DEDUCTION SYSTEM, VERIFICATION

General duration calculus is undecidable, and hence we cannot expect to find a complete set of axioms and deduction rules. The deductions in this paper may, hovewer, be based on the set of axioms and deduction rules given as follows.

A suitably decorated P, r or $\mathcal{D}$ in the formulas denotes a state assertion, a nonnegative real number or a formula in duration calculus. Symbols ff and tt are the truth values for state assertions. Note that general laws of the propositional and predicate calculus are not included, whereas some laws of interval temporal logic are.

Axiom 1: $\int ff = 0$

Axiom 2: $\int P \geq 0$

Axiom 3: $\int P_1 + \int P_2 = \int(P_1 \vee P_2) + \int(P_1 \wedge P_2)$

Axiom 4: If $r_1 \geq 0$ and $r_2 \geq 0$, then
$(\int P = r_1); (\int P = r_2) \Leftrightarrow \int P = (r_1 + r_2)$

Axiom 5: If $P_1 \Leftrightarrow P_2$ is a valid state assertion, then
$\int P_1 = \int P_2$ is an axiom.

The following induction rule is sound due to the finite variability of states.

Induction Rule: If $\mathcal{D}(\lceil\rceil)$ is deduced, and
$\mathcal{D}(X \vee (X; \lceil P\rceil) \vee (X; \lceil\neg P\rceil))$ is deducible from $\mathcal{D}(X)$, then $\mathcal{D}(true)$ is deduced.

It has a dual backward induction rule.

The following are rules of interval logic used in calculations.

Interval Law 1: Monotonic: If $\mathcal{D}_1 \Rightarrow \mathcal{D}_1'$ and $\mathcal{D}_2 \Rightarrow \mathcal{D}_2'$ then $\mathcal{D}_1; \mathcal{D}_2 \Rightarrow \mathcal{D}_1'; \mathcal{D}_2'$.

Interval Law 2: Associative:
$(\mathcal{D}_1; \mathcal{D}_2); \mathcal{D}_3 \Leftrightarrow \mathcal{D}_1; (\mathcal{D}_2; \mathcal{D}_3)$

Interval Law 3: False-Zero:
$false \Leftrightarrow \mathcal{D}; false \Leftrightarrow false; \mathcal{D}$

Interval Law 4: Point-Unit: $\mathcal{D} \Leftrightarrow \mathcal{D}; \lceil\rceil \Leftrightarrow \lceil\rceil; \mathcal{D}$

Interval Law 5: Chop-And:
$(\mathcal{D}_1; (\mathcal{D}_3 \wedge \ell = r)) \wedge (\mathcal{D}_2; (D_4 \wedge \ell = r))$
$\Rightarrow (\mathcal{D}_1 \wedge \mathcal{D}_2); (\mathcal{D}_3 \wedge D_4)$
$((\mathcal{D}_1 \wedge \ell = r); \mathcal{D}_2) \wedge ((\mathcal{D}_3 \wedge \ell = r); D_4)$
$\Rightarrow (\mathcal{D}_1 \wedge \mathcal{D}_3); (\mathcal{D}_2 \wedge D_4)$

Interval Law 6: Chop-Or:
$(\mathcal{D}_1 \vee \mathcal{D}_2); \mathcal{D}_3 \Leftrightarrow \mathcal{D}_1; \mathcal{D}_3 \vee \mathcal{D}_2; \mathcal{D}_3$
$\mathcal{D}_1; (\mathcal{D}_2 \vee \mathcal{D}_3) \Leftrightarrow \mathcal{D}_1; \mathcal{D}_2 \vee \mathcal{D}_1; \mathcal{D}_3$

Interval Law 7: Chop-Neg:
$\neg(\mathcal{D}_1; (\mathcal{D}_2 \wedge \ell = r))$
$\Leftrightarrow (true; (\neg\mathcal{D}_2 \wedge \ell = r)) \vee (\neg\mathcal{D}_1; \ell = r) \vee \ell < r$
$\neg((\mathcal{D}_1 \wedge \ell = r); \mathcal{D}_2)$
$\Leftrightarrow ((\neg\mathcal{D}_1 \wedge \ell = r); true) \vee (\ell = r; \neg\mathcal{D}_2) \vee \ell < r$

Interval Law 8: Exists-Chop:
$(\exists v : T \bullet \mathcal{D}_1); \mathcal{D}_2 \Leftrightarrow \exists v : T \bullet \mathcal{D}_1; \mathcal{D}_2$ provided v does not occur free in $\mathcal{D}_2$.

$\mathcal{D}_1; (\exists v : T \bullet \mathcal{D}_2) \Leftrightarrow \exists v : T \bullet \mathcal{D}_1; \mathcal{D}_2$ provided v does not occur free in $\mathcal{D}_1$.

The following derived laws have been useful:

Law 1: Dur-Range: $0 \leq \int P \leq \ell$

Law 2: Dur-Negation: $\int\neg P = \ell \Leftrightarrow \int P = 0$

Law 3: Dur-Chop-Add: Given a predicate over reals $R(r_1, \ldots, r_m)$, which is preserved under addition, i.e., $R(r_1, \ldots, r_m) \wedge R(r_1', \ldots, r_m') \Rightarrow R(r_1 + r_1', \ldots, r_m + r_m')$, we have
$R(\int P_1, \ldots, \int P_m); R(\int P_1, \ldots, \int P_m)$
$\Rightarrow R(\int P_1, \ldots, \int P_m)$

Law 4: State-Variation:
$\lceil\rceil \vee (\lceil P\rceil; true) \vee (\lceil\neg P\rceil; true)$
$\lceil\rceil \vee (true; \lceil P\rceil) \vee (true; \lceil\neg P\rceil)$

Law 5: State-And: $\lceil P_1 \wedge P_2\rceil \Leftrightarrow \lceil P_1\rceil \wedge \lceil P_2\rceil$

Law 6: State-Or: $\lceil P_1\rceil \vee \lceil P_2\rceil \Rightarrow \lceil P_1 \vee P_2\rceil$

Law 7: State-Negation: $\lceil\neg P\rceil \Rightarrow \neg\lceil P\rceil$

Law 8: State-Imply: $\lceil P_1 \Rightarrow P_2\rceil \Rightarrow (\lceil P_1\rceil \Rightarrow \lceil P_2\rceil)$

Law 9: State-Always: $\lceil P\rceil \Rightarrow \Box(\lceil P\rceil \vee \lceil\rceil)$

Law 10: State-Chop-And:
$\lceil P\rceil \wedge (\lceil P_1\rceil; \lceil P_2\rceil) \Leftrightarrow (\lceil P\rceil \wedge \lceil P_1\rceil); (\lceil P\rceil \wedge \lceil P_2\rceil)$

Law 11: State-Chop: $\lceil P\rceil; \lceil P\rceil \Leftrightarrow \lceil P\rceil$

ACKNOWLEDGMENT

We wish to express our sincere gratitude to our ProCoS colleagues, especially Prof. D. Bjørner and Prof. C. A. R. Hoare, for encouragement, support and useful discusions, and to Prof. Z. Chaochen for his continuing efforts to develop and adapt duration calculus to our use. Furthermore we have had useful discussions with Prof. J. Madey about the approach.

REFERENCES

[1] J. F. Allen, "Maintaining knowledge about temporal intervals," *Commun. Ass. Comput. Mach.*, vol. 26, pp. 832–843, 1983.
[2] ——, "Towards a general theory of action and time," *Artificial Intell.*, vol. 23, pp. 123–154, 1984.
[3] D. Bjørner, "A ProCoS project description," ESPRIT BRA 3104, *EATCS Bull.*, no. 39, Oct. 1989.
[4] A. M. Davis and P. A. Freeman, "Guest editors' introduction: Requirements engineering," *IEEE Trans. Software Eng.*, vol. SE-17, 3, pp. 210–211, 1991.
[5] J. Dawes, *The VDM-SL reference guide*. Pitmann, 1991.
[6] M. Degl'Innocenti, G. L. Ferrari, G. Pacini, and F. Turini, "RSF: A formalism for executable requirement specifications," *IEEE Trans. Software Eng.*, vol. 16, no. 11, pp. 1235–1246, 1990.
[7] R. C. Dorf, *Modern Control Systems, Addison-Wesley Series in Electrical Engineering*, 3rd ed. Reading, MA: Addison-Wesley, 1980.
[8] C. Ghezzi, D. Mandrioli, and A. Morzenti, "TRIO, a logic language for executable specifications of real-time systems," *J. Syst. Software*, vol. 12, no. 2, 1990.
[9] A. Goswami, M. Bell, and M. Joseph, "ISL: An interval logic for the specification of real-time programs," in *Proc. 2. Int. Symp. Formal Techniques in Real-Time and Fault-Tolerant Systems*, J. Vytopil, Ed., *LNCS* 571. New York: Springer-Verlag, 1991, pp. 1–20.
[10] R. Hale, "Temporal logic programming," in *Temporal Logic and Their Applications*, A. Galton, Ed. New York: Academic, 1987, pp. 91–119.
[11] A. G. Hamilton, *Logic for Mathematicians*, rev. ed. New York: Cambridge University Press, 1988.
[12] K. L. Heninger, "Specifying software Rrequirements for complex systems: New techniques and their application, *IEEE Trans. Software Eng.*, vol. SE-6, 1, pp. 2–13, 1980.
[13] K. M. Hansen, A. P. Ravn, and H. Rischel, "Specifying and verifying requirements of real-time systems," in *Proc. ACM SIGSOFT '91 Conf. On Software for Critical Systems*, New Orleans, LA, Dec. 4–6, 1991; *ACM Software Engineering Notes*, vol. 15, no. 5, pp. 44–54, 1991.
[14] M. R. Hansen and Z. Chaochen, "Semantics and Completeness of duration calculus," in *Proc. Real-Time: Theory in Practice, REX Workshop*, Mook, The Netherlands, June 1991, LNCS 600, 1992, pp. 209–225.
[15] D. Harel, "Statecharts: A visual formalism for complex systems," *Sci. Comp. Prog.*, vol. 8, pp. 231–274, 1987.
[16] C. A. R. Hoare, *Communicating Sequential processes*. Englewood Cliffs, NJ: Prentice-Hall, 1985.
[17] A. Isidori, *Nonlinear Control Systems*, Communications and Control Engineering Series. New York: Springer-Verlag, 1989.
[18] M. S. Jaffe, N. G. Leveson, M. P. E. Heimdahl, and B. E. Melhart, "Software requirements analysis for real-time process-control systems," *IEEE Trans. Software Eng.*, vol. SE-17, no. 3, pp. 241–258, 1991.

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on November 27, 2009 at 04:43 from IEEE Xplore. Restrictions apply.

[19] F. Jahanian and A. K.-L. Mok, "Safety analysis of timing properties in real-time systems," *IEEE Trans. Software Eng.*, vol. SE-12, 9, pp. 890–904, 1986.
[20] He Jifeng and J. Bowen, "Time interval semantics of a real-time programming Language, in *Proc. 4th Euromicro Workshop on Real-Time Systems*, Athens, Greece, June 3–5, 1992, pp. 110–115.
[21] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-Time Systems*, vol. 2, no. 4, pp. 255–299, 1990.
[22] L. Ljung, *System Identification. Theory for the User.* Englewood Cliffs, NJ: Prentice-Hall, 1987.
[23] D. G. Luenberger, *Introduction to Dynamic Systems. Theory, Models & Applications.* New York: Wiley, 1979.
[24] Luqui, V. Berzins, and R. T. Yeh, "A prototyping language for real-time software," *IEEE Trans. Software Eng.*, vol. 14, no. 10, pp. 1409–1423, 1988.
[25] K. Marzullo, "Tolerating failures of continuous-valued sensors," Tech. Rep. TR90–1156, Dept. of Computer Science, Cornell University, Ithaca, NY, Sept. 1990.
[26] O. Maler, Z. Manna, and A. Pnueli, "From timed to hybrid systems," in *Proc. Real-Time: Theory in Practice, REX Workshop*, Mook, The Netherlands, June 1991, LNCS 600, 1992, pp. 447–484.
[27] "The procurement of safety critical software in defence equipment; Part 1: Requirements, The procurement of safety critical software in defence equipment; Part 2: Guidance," Tech. Rep. INT DEF STAN 00–55, Ministry of Defence, Directorate of Standardization, Glasgow, Scotland, Apr. 1991.
[28] "Hazard analysis and safety classification of the computer and programmable electronic system elements of defence equipment," Tech. Rep. INT DEF STAN 00–56, Ministry of Defence, Directorate of Standardization, Glasgow, Scotland, Apr. 1991.
[29] B. Moszkowski, "A temporal logic for multilevel reasoning about hardware," *IEEE Trans. Comput.*, vol. C-18, 2, pp. 10–19, 1985.
[30] X. Nicollin, J. Sifakis, and S. Yovine, "From atp to timed graphs and hybrid systems," Draft Tech. Rep., June 1991.
[31] E.-R. Olderog, "Toward a design calculus for communicating programs," *Concur '91, 2nd International Conference on Concurrency Theory*, Amsterdam, The Netherlands, Aug. 1991, *LNCS* 527, 1991, pp. 61–77.
[32] D. L. Parnas and P. C. Clements, "A rational design process: How and why to fake it, "*IEEE Trans. Software Eng.*, vol. SE-12, no. 2, pp. 251–257, 1986.
[33] D. L. Parnas, G. J. K. Asmis, and J. Madey, "Assessment of safety-critical software," Tech. Rep. 90–295, TRIO, Queen's University, Kingston, Ontario, Canada, Dec. 1990.
[34] D. L. Parnas and J. Madey, "Functional documentation for computer system engineering" (version 2), CSL Rep. 237, TRIO, McMaster University, Hamilton, Ontario, Canada, Sept. 1991.
[35] A. Pnueli and E. Harel, "Applications of temporal logic to the specification of real-time systems" (extended abstract), in *Proc. Symp. on Formal Techniques in Real-Time and Fault-Tolerant Systems, LNCS 331*, M. Joseph, Ed. New York: Springer-Verlag, pp. 84–98.
[36] A. P. Ravn, H. Rischel, and V. Stavridou, "Provably correct safety critical software," in *Proc. IFAC SAFECOMP'90*, London, England, Oct. 1990, pp. 13–29.
[37] A. P. Ravn and H. Rischel, "Requirements capture for embedded real-time Systems, in *Proc. IMACS-MCTS'91 Symp. Modeling and Control of Technological Systems*, vol. 2, Villeneuve d'Ascq, France, 1991, pp. 147–152.
[38] G. M. Reed and V. W. Roscoe, "Metric spaces as models for real-time concurrency," *Mathematical Foundations of Programming, LNCS 298*, pp. 331–343, 1987.
[39] A. Saeed, R. de Lemos, and T. Anderson, "The role of formal methods in the requirements analysis of safety-critical systems: A train set example," in *Proc. 21st Symp. on Fault-Tolerant Computing*, 1991, pp. 478–485.
[40] F. B. Schneider, "The state machine approach: A tutorial," *Computing Surveys*, vol. 22, no. 3, 1990.
[41] S. Schneider, "Correctness and communication of real-time systems," Ph.D. dissertation, 1989, Tech. Monograph PRG-84, Oxford Univ. Computer Laboratory, England, March 1990.
[42] R. L. Schwartz, P. M. Melliar-Smith, and F. H. Vogt, "An interval logic for higher-level temporal reasoning," in *Proc. 2nd Annual ACM Symp. on Principles of Distributed Computing*, 1983, pp. 173–186.
[43] A. C. Shaw, "Reasoning about time in higher-level language software," *IEEE Trans. Software Eng.*, vol. 15, no. 7, pp. 875–889, 1989.
[44] J. M. Spivey, *The Z Notation.* Prentice-Hall International Series in Computer Science. Englewood Cliffs, NJ: Prentice–Hall, 1989.
[45] E. V. Sørensen, A. P. Ravn, and H. Rischel, "Control program for a gas burner: Part 1: Informal requirements, ProCoS case study 1," ProCoS Rep. ID/DTH EVS2, 1990.
[46] J. C. Willems, "Paradigms and puzzles in the theory of dynamical systems," *IEEE Trans. Automat. Contr.*, vol. 36, no. 3, pp. 259–294, 1991.
[47] Z. Chaochen, C. A. R. Hoare, and A. P. Ravn, "A calculus of durations," *Information Processing Lett.*, vol. 40, no. 5, pp. 269–276, 1991.
[48] Z. Chaochen, M. R. Hansen, A. P. Ravn, and H. Rischel, "Duration specifications for shared processors," in *Proc. 2nd Int. Symp. on Formal Techniques in Real-Time and Fault-Tolerant Systems, LNCS 571*, J. Vytopil, Ed., 1991, pp. 21–32.

Anders P. Ravn (M'83) received the M.S. degree in computer science from the University of Copenhagen, Denmark, in 1973.

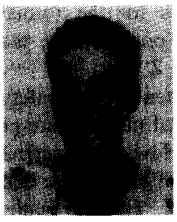
He is Associate Professor with the Department of Computer Science, Technical University of Denmark, Lyngby, Denmark. His research interests are in the areas of software engineering principles for embedded computing systems and specification of total systems in real-time applications.

Prof. Ravn is a member of the Association for Computing Machinery.

Hans Rischel (M'91) received the M.S. degree in mathematics from the University of Copenhagan, Denmark, in 1960.

He was Assistant and Associate Professor with the Department of Mathematics, University of Copenhagen, from 1962 to 1970. From 1970 to 1984 he was employed in industry. Since 1985 he has been Associate Professor with the Department of Computer Science, Technical University of Denmark. His research interest is in software engineering with special focus on the use of mathematical methods.

Prof. Rischel is a member of the Association of Computing Machinery.



Kirsten Mark Hansen received the M.S. degree in software engineering from the Department of Computer Science, Technical University of Denmark, Lynby, Denmark, in 1989.

Since 1989, she has been with the Technical University as a research assistant. Her research interest is in the area of software engineering, especially design and verification of safety-critical and embedded real-time systems.

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on November 27, 2009 at 04:43 from IEEE Xplore. Restrictions apply.